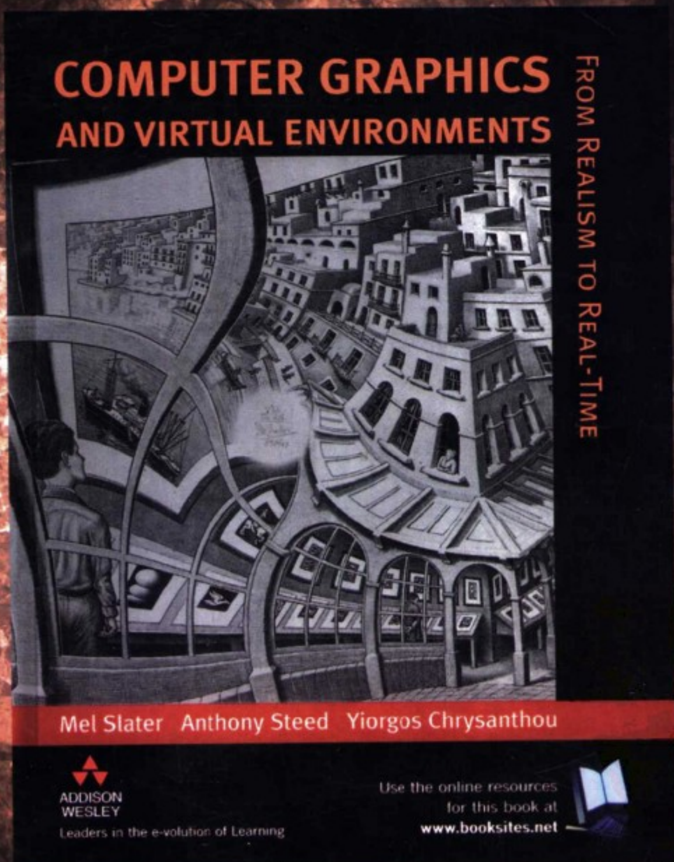
计 算 机 科 学 丛 书

计算机图形学与虚拟环境

（英）Mel Slater Anthony Steed Yiorgos Chrysanthou 著 程成 徐玉田 译



Computer Graphics and Virtual Environments From Realism to Real-Time

机械工业出版社
China Machine Press

VISIT…

LANZAROTE Caliente.com

实时虚拟环境 (VE) 在信息革命中占有重要地位。本书介绍了如何用“自顶向下”方法构造实时图形系统。本书从与VE密切相关的感知和光照问题开始，之后介绍了如何用光线跟踪来产生高质量的图像。通过逐步放松对光线跟踪的假设限制，逐章讲解实时生成三维图像的图形系统。本书特点是，为学生讲授一些实用知识，包括图形渲染管道的基本原理、使帧频最大化的最前沿技术以及渲染具有照片真实感的图像的深层内容，从而激发学生围绕虚拟环境进行深入讨论。

作者简介

Mel Slater 伦敦大学学院 (University College London, UCL) 计算机科学系虚拟环境领域的教授。他曾经是加州大学伯克利分校的访问教授和麻省理工学院 (MIT) 电子学研究实验室的访问学者。他是《Presence: Teleoperators and Virtual Environments》杂志的高级编辑，《Computer Graphics: Systems and Concepts》一书的合著者（此书的另一位作者是Rod Salmon），也是《Distributed Windows Systems》一书的合著者。他目前是EPSRC的高级研究员，致力于研究计算机图形学的光域方法。

Anthony Steed 伦敦大学学院计算机科学系虚拟环境领域的讲师。主要研究方向是沉浸式虚拟环境、分布式虚拟现实系统可扩展性支持以及虚拟空间中参与者的合作。

Yiorgos Chrysanthou 伦敦大学学院计算机科学系虚拟环境领域的讲师。主要研究方向是实时渲染、布料建模、仿真和计算几何。

ISBN 7-111-14824-X

9 787111 148241

封面设计 陈子平

HZ BOOKS
华章图书

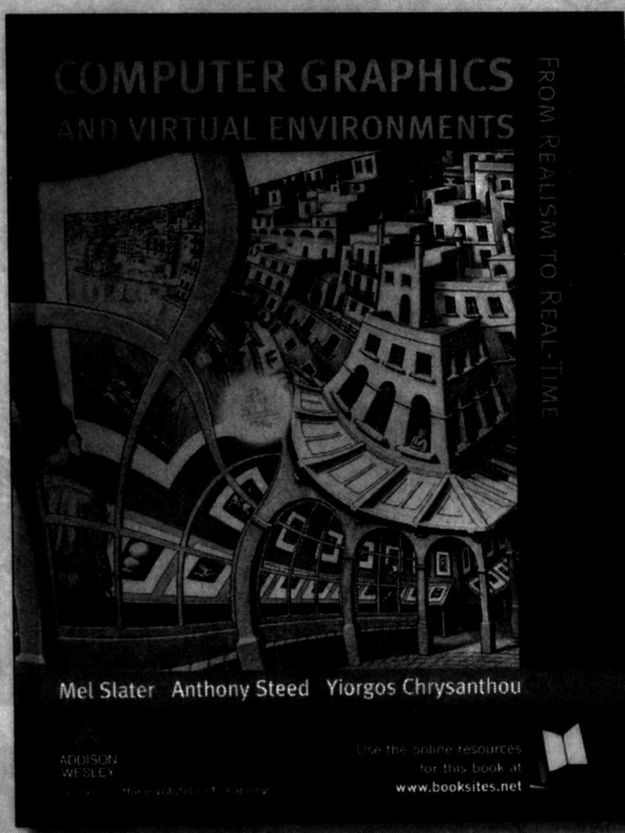
华章网站 http://www.hzbook.com
网上购书：www.china-pub.com
北京市西城区百万庄南街1号 100037
读者服务热线：(010)68995259，68995264
读者服务信箱：hzedu@hzbook.com

ISBN 7-111-14824-X/TP · 3510
定价：59.00 元

计算机科学丛书

计算机图形学与虚拟环境

（英）Mel Slater　Anthony Steed　Yiorgos Chrysanthou 著　　程成　徐玉田 译



Computer Graphics and Virtual Environments From Realism to Real-Time

机械工业出版社
China Machine Press

本书以“自顶向下”的方式阐述了如何构造实时图形系统，并用相当篇幅介绍了虚拟环境的感知问题、动态特性和交互，以及与此相关的显示和交互设备方面的问题。本书围绕虚拟环境展开对图形学理论和技术的介绍，在内容编排和组织上独具特色。

本书适合用作学习计算机图形学及虚拟环境相关课程的大学高年级本科生或研究生的教材，同时也可供相关技术人员阅读。

Mel Slater, Anthony Steed, Yiorgos Chrysanthou: Computer Graphics and Virtual Environments : From Realism to Real-Time (ISBN:0-201-62420-6).

Copyright © 2002 by Pearson Education Limited.

This translation of *Computer Graphics and Virtual Environments : From Realism to Real-Time* (ISBN:0-201-62420-6) is published by arrangement with Pearson Education Limited.

All rights reserved.

本书中文简体字版由英国Pearson Education培生教育出版集团授权出版。

版权所有，侵权必究。

本书版权登记号：图字：01-2002-3591

图书在版编目（CIP）数据

计算机图形学与虚拟环境/（英）斯拉特（Slater, M.）等著；程成、徐玉田译．－北京：机械工业出版社，2004.10

（计算机科学丛书）

书名原文：Computer Graphics and Virtual Environments : From Realism to Real-Time

ISBN 7-111-14824-X

Ⅰ．计…　Ⅱ．①斯…　②程…　③徐…　Ⅲ．计算机图形学　Ⅳ．TP391.41

中国版本图书馆CIP数据核字（2004）第064880号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：刘　渊

北京中兴印刷有限公司印刷 · 新华书店北京发行所发行

2004年10月第1版第1次印刷

787mm × 1092mm 1/16 · 29(彩插1印张)印张

印数：0 001-4 000册

定价：59.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到"出版要为教育服务"。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以"计算机科学丛书"为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

"计算机科学丛书"的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，"计算机科学丛书"已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在"华章教育"的总规划之下出版三个系列的计算机教材：除"计算机科学丛书"之外，对影印版的教材，则单独开辟出"经典原版书库"；同时，引进全美通行的教学辅导书"Schaum's Outlines"系列组成"全美经典学习指导系列"。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成"专家指导委员会"，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘 书 组

武卫东　温莉芳　刘　江　杨海玲

译 者 序

经过三个多月的努力，我们终于拿出了本书的中文译稿，此时的心情非常轻松和愉悦。往日的辛劳没有让我们感到任何痛苦，厚达近600页的原书中，每一页都有如一杯浓香的咖啡令人回味。如果要说最大的感触，我想可能就是先睹为快的感觉，以及能做点儿事让同行们一同分享该书的快乐了。

本书的特色之一就是没有泛泛介绍图形学的各方面细节内容，而是紧紧围绕图形学最为重要也是未来最广泛的应用形式——“虚拟环境”展开对图形学理论和技术的阐述。本书的第二个特色便是在内容编排和组织上真正做到了深入浅出，既能够保证读者在开始就掌握图形学理论框架并建立起图形学试验环境和平台，又有相当深刻的内容介绍虚拟环境中所面临的技术困难，使读者了解如何能够在真实和实时之间鱼和熊掌兼得。第三个特色，也是本书最为闪光之处，在于作者将图形学相关内容十分完美地整合在一个理论框架内。这一点显示出作者在图形学理论研究上的深厚功力，同时对于读者深刻理解图形学理论和方法有极大的帮助。本书的第四个特色是作者摆脱了图形学传统教科书的写作视角，面向虚拟环境，融合了认知心理学和人机交互的相关内容，以全新的视点和宽阔的视域阐述计算机图形学的理论和技术。本书内容丰富，又有相当的深度，既可以挑选通用内容作为大学本科生教材，也可以选择有深度的内容作为图形学和人机交互领域的研究生和博士生的教科书。

本书由程成和徐玉田翻译，由于译者自身的知识局限及时间仓促，译稿中难免有错误和遗漏，谨向原书作者及读者表示歉意，并欢迎批评指正。

译 者

2004年5月

前言

通信技术在过去十多年间发生了巨大的变化，通信技术水平日益提高。实时虚拟环境正是这种技术革命带来的一个重要成果。我们可以观看由虚拟新闻播音员播讲的新闻，可以同分布在世界各地的游戏玩家一起玩3D图形游戏，也可以在虚拟环境中和虚拟银行经理就贷款事宜进行商谈。计算机图形技术为我们创造了虚拟环境，另一方面，随着我们对实时虚拟环境系统需求的不断增加，计算机图形学也得以不断发展。同时，我们也希望虚拟环境能提供更强的真实感。计算机图形学要在这样的需求冲突中不断向前发展。这本书是与实时虚拟环境紧密相关的计算机图形学著作，我们的目标是将其做成实时虚拟环境图形学导论教材。本书介绍如何构建实时图形系统、为此需要牺牲哪些真实感。与此同时用相当篇幅介绍了虚拟环境的感知问题、动态特性和交互、以及与此相关的显示设备和交互设备方面的问题。

作为一本3D计算机图形学和虚拟环境的教科书，本书在很多方面是与众不同的。本书开篇在较高的层次上阐述了与虚拟环境、光照、颜色感知相关的感知问题。其次，为了更好地理解书中的理论，一些需要掌握的数学概念和知识也在前面章节中加以介绍。然后立即转向非常实际的应用研究、讲解使用光线跟踪技术轻松实现全局光照条件下具有照片真实感的高质量图像的最流行方法。之后逐渐向实时系统推进，通过不断放松对光线跟踪的假设，一步步直到实现一个能在实时条件下生成3D场景图像的系统。换句话说，我们是在描述图形系统典型的渲染管道。传统的计算机图形是以点光源为基础的，这种点光源至多可以产生看起来很“硬”的阴影。而在实时系统的环境中，为了获得更强的光照真实感，我们展示“柔和”的、看起来更真实的阴影的产生过程。然后再回到使用辐射度算法生成全局光照条件下具有照片真实感的图像的问题，该算法也在特定情况下使用了实时渲染。在此之后，我们再一次讨论光线跟踪，介绍多种能有效提升光线跟踪速度的技术(虽然它还不是真正的实时)。

这本书是真正按“自顶向下”方式来阐述的。举例来说，为了引出2D直线的裁剪和渲染，我们先阐述加速光线跟踪的必要方法，在这样的上下文中来分析该问题。（在统一细分空间中跟踪直线与在2D显示设备上渲染直线是相似的。）我们相信这种方式是合乎教学法的——2D直线的裁剪和渲染问题本身是不太有趣的，但是在比较高层次的操作中，例如光线跟踪或辐射度等，这些基本算法就显示出了生命力，在另一方面也生动地说明这些算法本身所包含的解决问题的方法是完全超越了其特定应用的。事实上，计算机图形学课程不应局限在介绍一些技术内容，更应该分析和讲解这些技术背后所包含的思维方式。

这本书将渲染和建模很好地分离开来，在介绍渲染的部分中，主要讨论了两种渲染方法；在介绍建模的部分中，主要阐述了体素构造表示、曲线和曲面建模技术。介绍完这两大部分内容之后，我们转向与虚拟环境直接相关的内容：虚拟世界里的动态特性和交互。

最后两章再一次探讨从真实到实时这个关键问题，不过，这次我们将站在更高的层次上来分析。其中一章综述全局光照的一些先进技术，例如路径跟踪和光子跟踪。另外一章回顾实时图形系统中提高帧频的一系列方法：基于可见性、细节层次上的技术和基于图像的渲染技术。这一章的最后部分介绍了一种新的称为“光域”的计算机图形学范式。

本书所包含的例子主要来自两大最为重要和最为流行的系统：场景描述语言VRML和实时渲染系统OpenGL。我们假设读者都熟悉C语言语法,并至少熟练掌握C、C++或Java语言的一种。本书中的所有例子都采用类C语言的记法。

不像许多其他的教科书，我们没有将数学基础的内容放在附录中，而是在书的前面部分开辟专门一章来介绍。具备大学计算机系本科生相应的数学知识是理解这本书的必要前提，因此，本书将这些数学基础内容放在第一部分第2章提前讲解，另外一些有关的数学内容则穿插在各章中介绍。不管读者初次翻看此书时会有什么样的感觉，书中并没有艰深的数学内容。如果被$\int_B\int_A f(x,y)dxdy$这样的积分公式吓住了，请不要害怕，实际上很简单，它表示的意思是这样的一个函数：第一步，在XY平面上的点(x,y)处求$f(x,y)$的值，取(x,y)点周围的一个很小区域$(dx \times dy)$乘以该值；第二步，在x的区间A和y的区间B所围成的平面区域的每一点上执行第一步计算，并将这些“体积”累加在一起。理解这本书并不需要读者了解该如何实际计算这样的积分。然而，了解类似上述表达式的含义是读者在一般层次上理解这本书所需要的。

这本书的读者群主要为高年级大学生（特别是计算机科学系最后一年的本科生）或那些研究计算机图形学、虚拟环境视觉问题的研究生。它可以作为整个学期（30～45个小时）课程的教材。本书的目标是要传授给学生在图形学和虚拟现实实际应用中所需掌握的知识，这些知识包括图形渲染管道的基本内容、使帧频最大化的最新技术，以及一些关于渲染具有照片真实感的图像的深层次内容。本书的一个焦点是虚拟环境。因此我们不只是介绍计算机图形学的算法,而且还把这些算法和一些基本的感知问题放在虚拟环境的上下文中讲解。

将本书作为大学本科教材的用法可以是：首先从第 5 章开始,在包含球体的简单场景上下文中介绍光线投射。其目的是从第一天起就让学生真正能参与到图形系统构造的实际过程当中，使之能够尽快创建光线跟踪的全局光照场景；然后讲述第1~4章的内容，其覆盖了感知的问题（第1章通过简单的感知实验让学生感受到学习这些内容的乐趣）；接下来复习第2章中的相关数学知识，因为不是每个人都掌握了这一章的数学基础知识；第3章的光照和光亮度方程的一般性问题是非常重要的内容，至少需要一两次讨论课的时间来完成这部分教学。因为光亮度方程（radiance equation）是计算机图形渲染的本质内容，也是有关渲染的所有各章内容的统一原理；而后继续讨论第4章的颜色感知；之后可以转到第6章的光线跟踪，布置关于光线跟踪程序的作业，该程序要求在包含球体的简单场景中完成光线跟踪。大约在三个星期内学生会生成光线跟踪的图像，这将是对他们信心的极大鼓舞。在此过程中学生基本了解了图形渲染的核心内容——光亮度方程。

接下来是第7~13章的内容，主要围绕如何建立一个实时系统展开论述。可以通过一些问题来激发学生的思维。光线跟踪为什么如此之慢？如何表现复杂场景？如何能对任意放置的虚拟照相机进行渲染？如果不用光线跟踪的全局光照特性，如何进行明暗处理？如果每个对象上的每个点都必须做明暗处理,如何使实时渲染成为可能？如果不用光线跟踪的自动可见性计算,如何计算一般场景中的可见性？在第13章结束之前，读者就会对建立一个完全的3D图形系统有了全面的认识、从如何设置某个像素为特定颜色这样一个基本功能开始，一步步完善，直至整个3D图形系统完成。因此虽然我们的叙述遵从“自顶向下”的方式，在最后它也是“自底向上”的，因为第7~13章的主要内容就是自底向上地建立3D图形系统。

某些类型的大学本科课程也可以直接从第15章开始,即用实例说明辐射度解(radiosity solution)是如何得到的；然后接着进入第16章，讲解光线跟踪，分析如何加速光线跟踪；最后

以第17章结束课程，该章分析最基本的一些图形算法，以及如何裁剪和渲染直线（我们在第16章讲解快速光线跟踪时提到过这些操作）。

在讲完第7~13章之后，可以接着进入一些深层次的内容，即第14~16章的实时阴影生成、辐射度(radiosity)以及快速光线跟踪。有关体素构造表示、曲线和曲面的计算机辅助几何设计是相对独立的部分。然后可以转到第22章和第23章，这两章讨论的是全局光照和实时渲染的高级问题。

如果读者对虚拟环境的交互问题、实时交互系统的一般问题（诸如碰撞检测等）也感兴趣的话，也可以抛开渲染部分来研究第20章和第21章两章的内容。

这本书最初的版本写于1991年，那时作者之一 (Mel Slater) 正在加州大学伯克利分校教授CS184(计算机图形学基础)。从那以后，本书被伦敦大学玛丽皇后学院和伦敦大学学院（UCL）用作教材。在玛丽皇后学院（1992~1995年），本书被用作交互式计算机图形学课程的教材。1995年之后，本书在伦敦大学学院作为计算机图形学的教材。其他版本也在兰开斯特大学和南非的开普敦大学使用。多所大学还把它用在硕士课程上。在加州大学伯克利分校（1992年），在玛丽皇后学院（1992~1995年）以及UCL，它用作硕士生计算机图形学和虚拟环境课程的教材。Mel Slater 于1995年从玛丽皇后学院来到UCL，和此时已经在UCL任教的他先前的两位博士生Anthony Steed和Yiorgos Chrysanthou 一起教授计算机图形学课程，这本书正是他们合作的结晶。

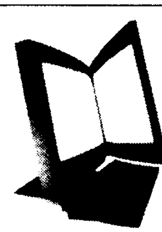
Mel Slater
Anthony Steed
Yiorgos Chrysanthou
2001年4月于伦敦

致 谢

如果没有很多人的协助，这本书就不可能完成。我们特别想感谢Martin Usoh、Tony Tsung-Yueh Lin 和 Edwin Blake，他们在较早阶段参与了本书的工作。Jesper Mortensen、Joao Oliveira、David-Paul Pertaub和 Franco Tecchia 在书中一些图的制作上给予了很多帮助。我们还想感谢Christian Babski、Mireille Clavien、Ioannis Douros、George Drettakis、Neil Gatenby 、David Hedley、Marc Levoy、Alf Linney、Céline Loscos、David Luebke、Alan Penn、Claudio Privitera、Bernard Spanlang、Lawrence Stark、Daniel Thalman、Tzvetomir Vassilev、Greg Ward、Mary Whitton、Peter Wonka 以及 Hansong Zhang，他们为书中的图收集材料。感谢Amy Goldstein所做的建模工作。我们还想感谢以下各位，他们都以各种不同的方式提供了帮助：Alan Chalmers、Daniel Cohen-Or、Rabin Ezra、Patrick M. Hanrahan、Phil Huxley、David Mizell、Jan-Peter Muller、Gareth Smith和 Shankar N. Swamy。

我们在此还要感谢下列公司和机构：瑞典斯德哥尔摩皇家工学院并行计算机中心 (PDC)、Light-works公司、SGI公司以及 Virtual Research Systems公司。特别要感谢Pearson Education公司的工作人员，尤其是Keith Mansfield，是他这些年以来一直帮助和鼓励我们。特别感谢资深编辑Mary Lince，是她协助我们完成了书稿的最后整理工作。

最后我们要感谢那些在伦敦大学学院、伦敦大学玛丽皇后学院和加州大学伯克利分校学习图形学的学生，他们是这项工作的灵感源泉。



Mel Slater、Anthony Steed、Yiorgos Chrysanthou为《计算机图形学与虚拟环境》一书提供了一个配套Web站点www.booksites.net/slater，敬请访问。

你将会发现很多有价值的参考资料，包括：

针对学生：

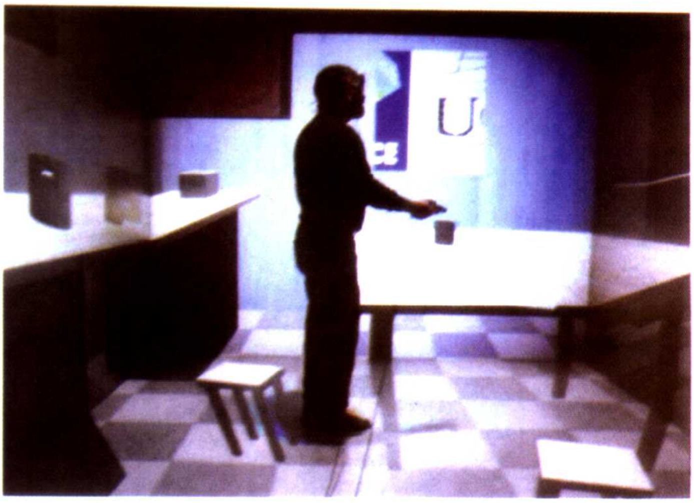
- 章节练习

针对教师：

- 教学用Powerpoint 幻灯片

针对学生和教师：

- 书中相关的类库，包括运行在Windows、Irix和 Linux 系统上的 C 语言和 Java 语言的类库



彩图1-3 虚拟厨房中的人，用Trimension ReaCTor（一种类似于CAVE的系统）显示



彩图1-7 虚拟千禧眼（由Rick Mather Architects 提供）

彩图1-8　Maitreya 项目 (Maitreya Project International有限公司版权所有，http://www.maitreyaproject.org)

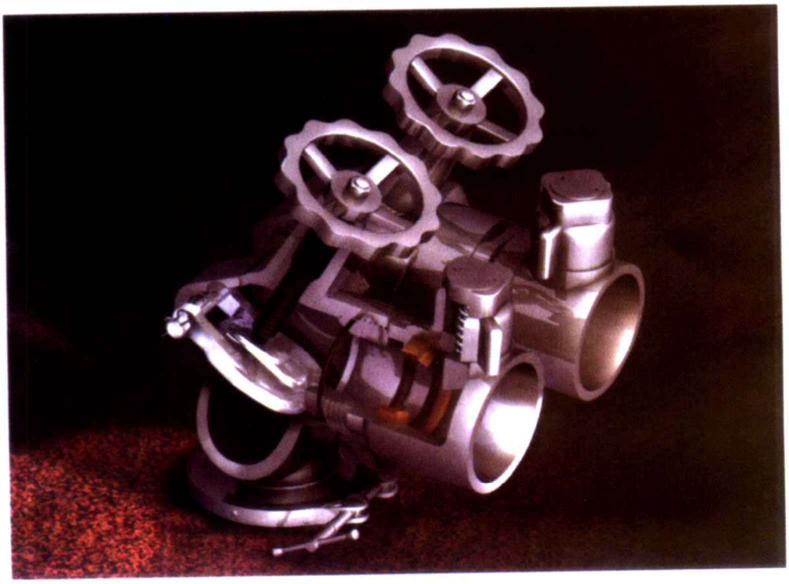


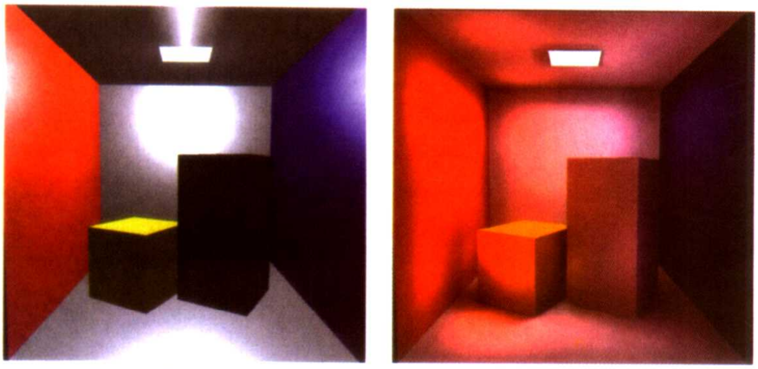
a)



b)

彩图1-9　排气阀，使用Z型RenderZone和集成LightWorks渲染器生成。这些消防泵排气阀是Real-Time Visualisation公司操作员培训指导中动画展示的一部分（Real-Time Visualisation公司版权所有）





a) 局部光照渲染的场景　　b) 全局光照渲染的场景

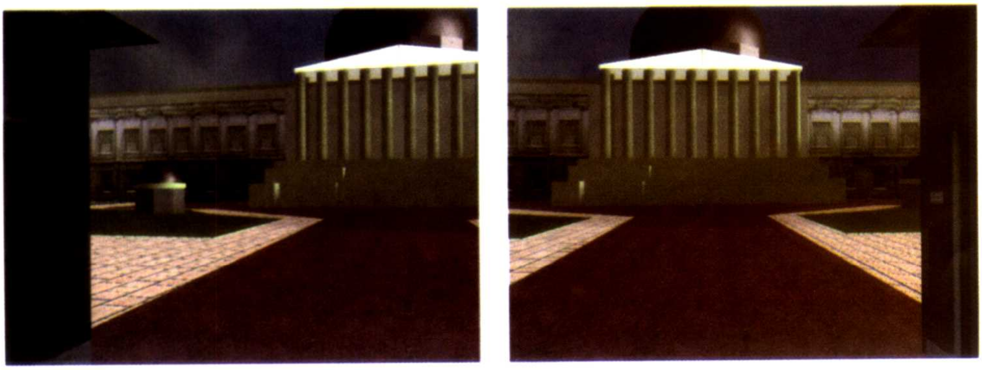
彩图1-10　Cornell 方盒（由UCL计算机科学系的Jesper Mortensen 提供）



彩图1-11　面向公众讲话时的紧张——纹理映射的一个实例（由UCL计算机科学系的David-Paul Pertaub提供）



彩图1-13 《繁星之夜》，1888年。作者梵高（由Lauros-Giraudon /Bridgeman艺术图书馆提供）



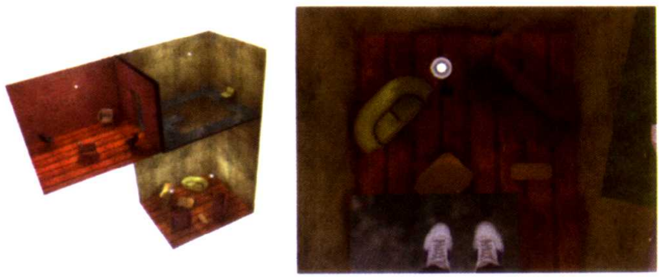
彩图1-16 UCL的虚拟模型



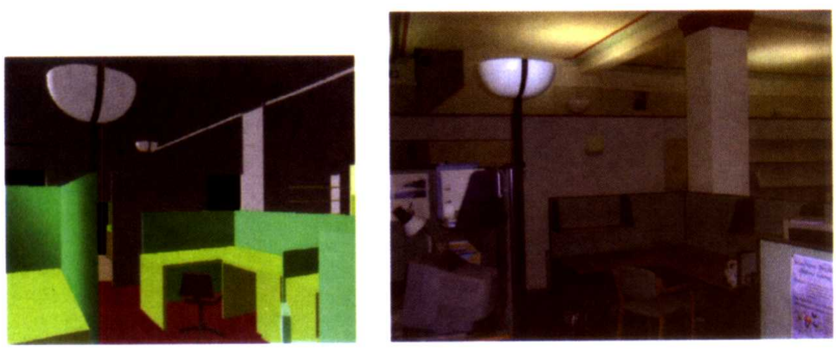
彩图1-17　物体布局（经ACM许可使用，1999年）



彩图1-18　布料行为的仿真（由UCL计算机科学系的Bernhard Spanland和Tzvetomir Vassilev 提供）



彩图1-20 Pit Room[UNC, UCL Experiment]（经ACM许可使用，1999年）



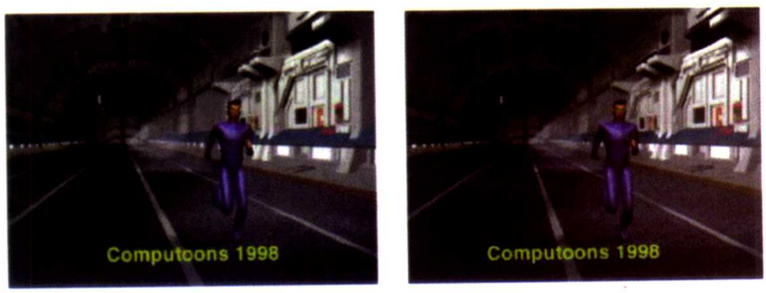
a）虚拟实验室

b）所对应的真实实验室

彩图1-21 实验室的虚拟和真实视图

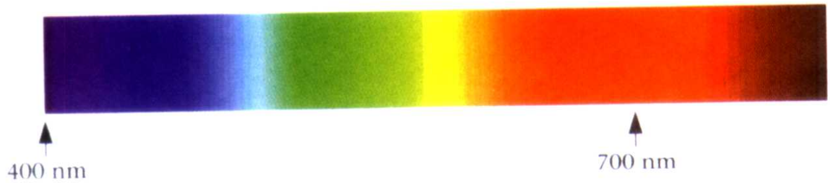


彩图1-23 Ames Room（由Exploratorium提供）

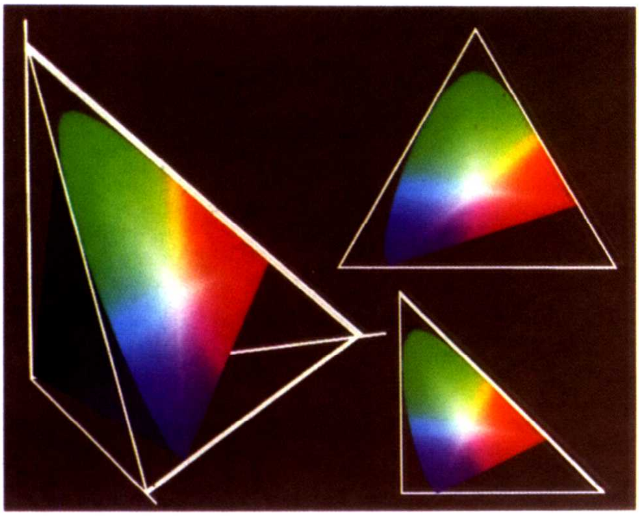




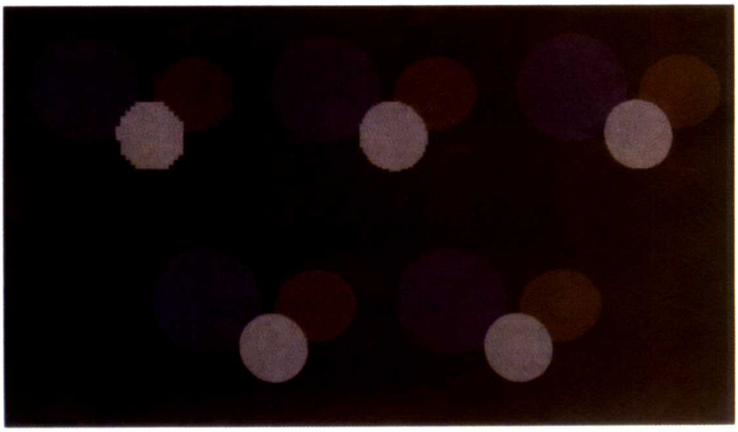
彩图1-33　立体对(“奔跑者”由Computoons公司提供，“商店场景”由O.W. Holmes 立体视觉研究实验室提供，http:// www.stereoview.org)



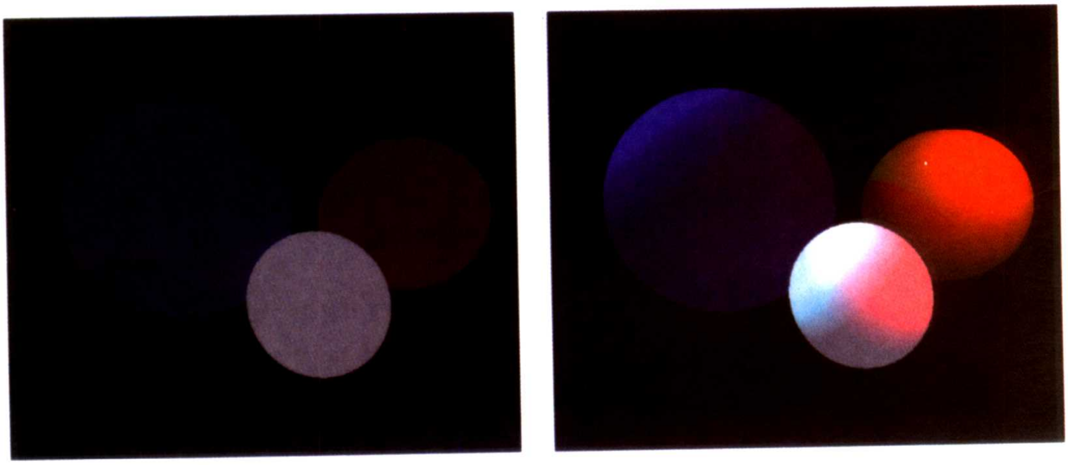
彩图4-3　可见光谱



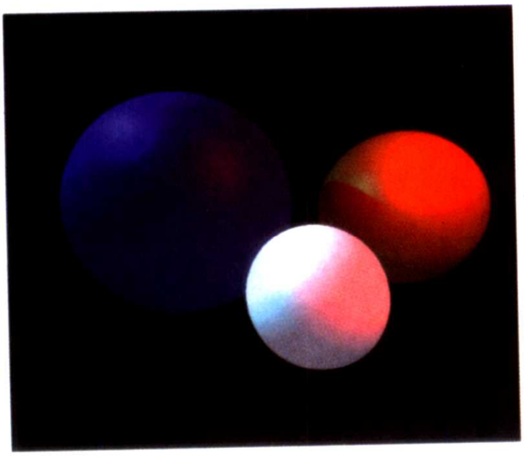
彩图4-9　CIE-XYZ色度图
(由Barbara Meier 提供)



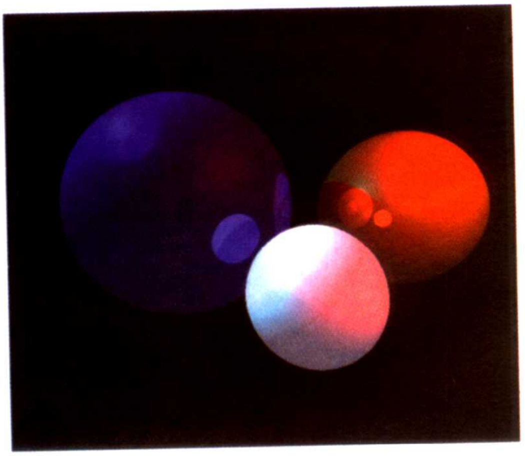
彩图5-6　一组分辨率逐渐提高的图像



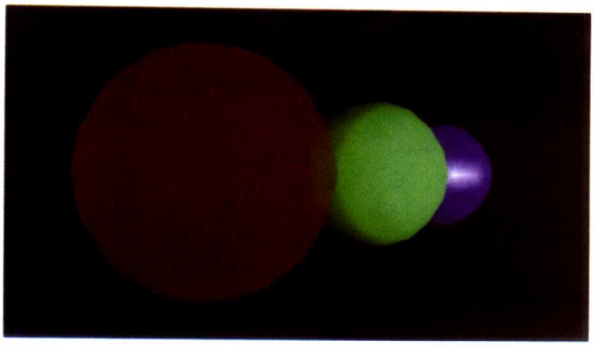
彩图6-9 利用漫反射的光线投射



彩图6-10 利用镜面反射的光线投射



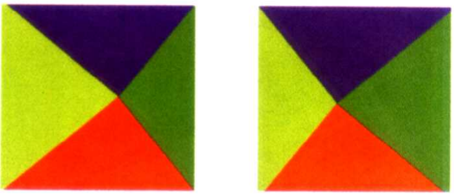
彩图6-12 光线跟踪实例——不透明表面



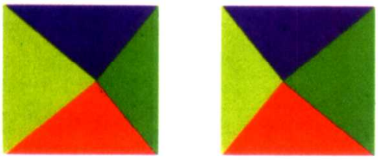
彩图6-15 b) VRML97 材质示例



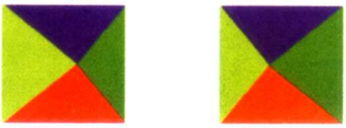
彩图9-12 从顶部观察一个棱锥



彩图9-14 立体视图，视平面距离为100（VP 位于棱锥的后面）

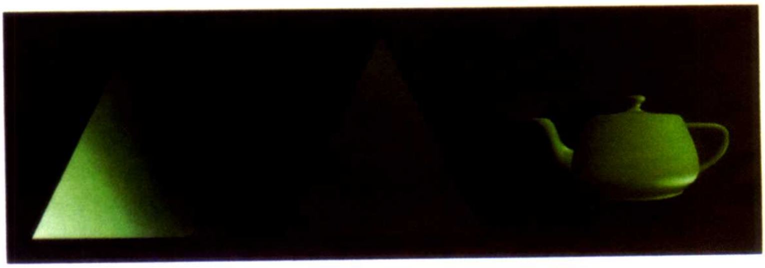


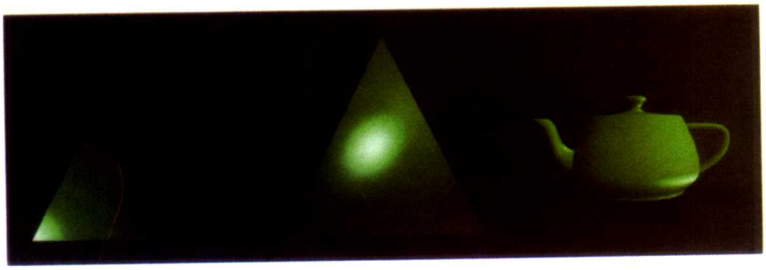
彩图9-15 立体视图，视平面距离为80（VP与棱锥相交）



彩图9-16 立体视图，视平面距离为60（VP与棱锥相交）

彩图13-3 Gouraud 明暗处理的图例（由UCL计算机科学系的Jesper Mortensen和Alican Met提供）

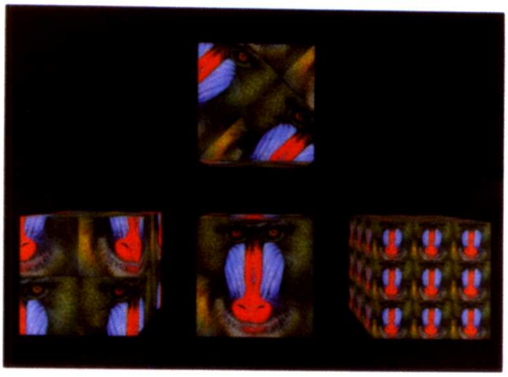




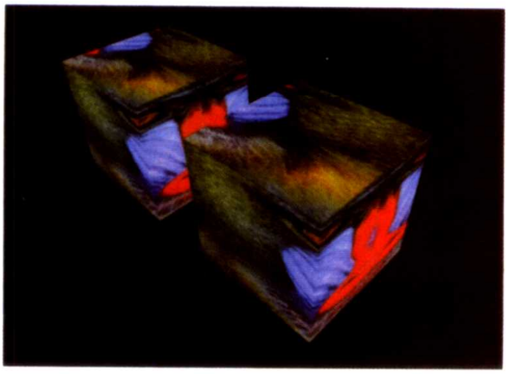
彩图13-4 Phong 明暗处理的图例（由UCL计算机科学系的Jesper Mortensen 和Alican Met 提供）



彩图13-13 纹理堆叠的立方体实例



彩图13-14 b) 纹理映射的基本体素实例



彩图13-15 b) 纹理映射的IndexedFaceSet实例



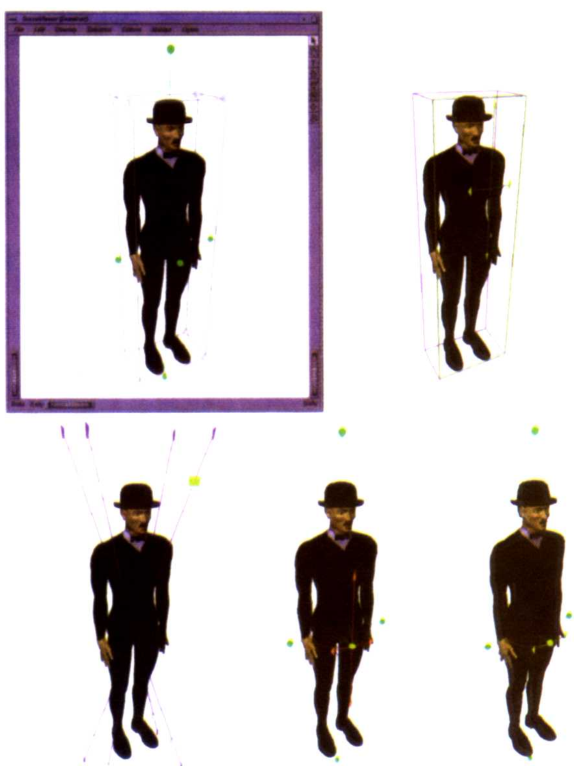


彩图14-19 (左) 加入1000个多边形的场景。只有直接光照，需要CPU 29.3分钟（在SCI Indigo 2 R4000上运行）用于不连续网格化、背面投影和光照计算，计算使用Drettakis-Fiume 算法（右）相同的场景，只是将不连续网格叠置其上

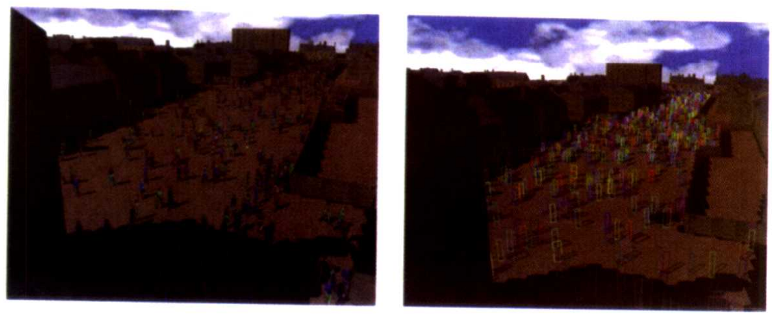
（由法国iMAGIS-GRAVIR公司的George Drettakis 提供）



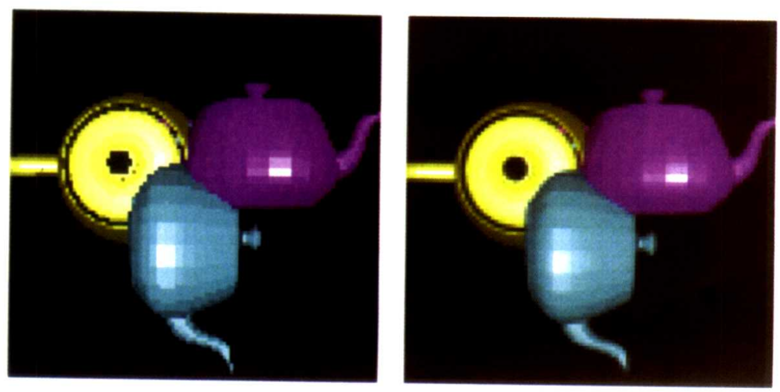
彩图20-2　位于瑞典斯德哥尔摩皇家工学院(KTH)并行计算机中心（PDC）的VR-CUBE



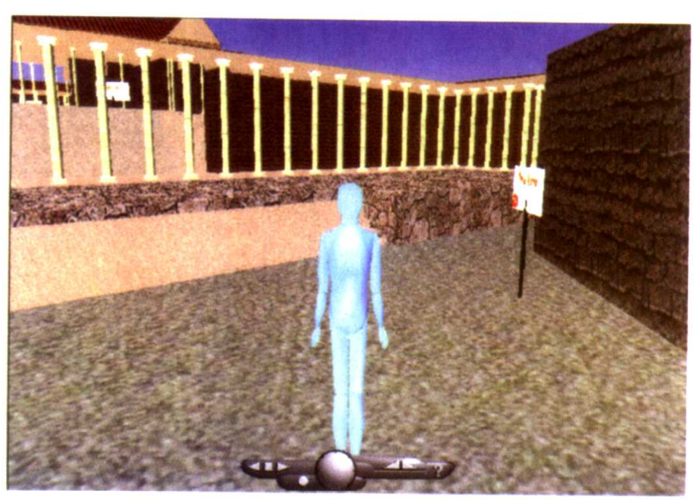
彩图21-6　SceneViewer中的变换操纵器（人体模型由瑞士联邦技术研究所（EPFL）的计算机图形学实验室提供）



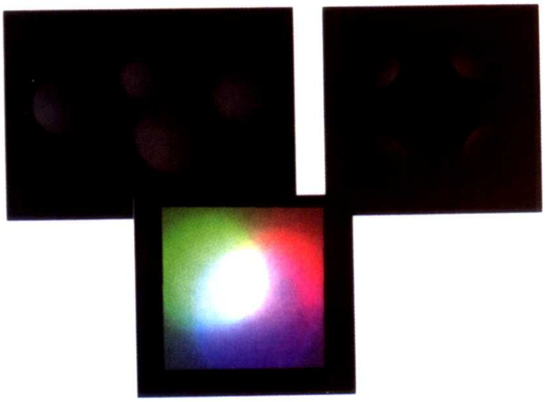
彩图23-16 大场景下基于图像的渲染实例（由UCL计算机科学系的Franco Tecchia提供）



彩图23-25 全屏幕反走样



彩图A-1 VRML 世界示例



彩图A-4 DirectionalLight 、PointLight 和SpotLight示例



彩图P-1 明亮的办公室：有家具的房间内部视图
（Greg Ward版权所有，1997年）



彩图P-2　利用Light-Works 辐射度方法，建筑师能模拟日光穿过窗子照射进来的场景。另外，系统为复杂几何体模拟出光照效果，比如曲面和场景中光照层次的变化



彩图P-3　米开朗基罗的雕塑《夜》的光域（由斯坦福大学“数字米开朗基罗”项目的Marc Levoy教授提供，http://graphics.stanford.edu/projects/mich/lightfield-of-night/lightfield-of-night. html）

第一部分　绪论：感知、光、颜色和数学

第1章　导论：投影的虚幻世界

"什么是真正的世界，Don Juan？"

"一个产生能量的世界，与投影的虚幻世界相对的概念。在虚幻世界中，就如同我们大多数人所做的梦一样，没有能量效果。"

Carlos Casteneda，《梦的艺术》

HarperCollins 出版社，纽约，1993年，第164页

1.1　引言

计算机图形学是有关建模、光照、虚拟世界的动态特性以及人们在其中的活动方式的理论。它产生一个"投影的虚幻世界"。这个所谓的"虚幻世界"今天被称作"虚拟环境"或"虚拟现实"。这是"投影世界"的另一种更富文学色彩的名字。所谓"投影世界"即是将三维空间中的对象表示投影到二维显示器上所形成的表现。人们看到这样的二维显示器中的图形所形成的虚幻对象并与之交互，从中体验一种虚幻上的真实。此时人们的行为表现仿佛人们就存在于这个虚假的世界中，而周围真正的世界却变得遥远和暗淡了。人们在非真实的环境中的行为本身揭示和增强了人们的心理模型，这个心理模型正是我们对现实产生的幻觉的重要内容。 1

储存抽象表示并运行程序的计算机以及最后产生图像的二维硬件可能通过网络联接到其他的计算机上，在网络的那一端，人们将会看到同一个虚幻世界的不同侧面。参与这个虚拟现实的人们本身也被表现为虚拟现实中的一个三维对象，然后被投影到二维显示器上，这样分布在不同地点的人们就可以相互看见"对方"并彼此交互。因此虚拟现实变成了一种共享的和社会性的现实。

如此的网络虚拟环境今天已经存在，这种网络虚拟社区具有无限的疆域，容许数以千计的人共同参与（Singhal and Zyda，1999）。网络虚拟环境是多领域技术的集成，这些技术包括允许不同地域间异质网络互联的技术，支持视觉、听觉、触觉/力觉反馈技术以及相应的多种交互设备。

1.2　范围

这本书与虚拟环境的视觉方面有关——创建、光照、真实地显示三维的虚拟环境，并且允许人们与之交互。

我们采用与众不同的一种论述方式，即"自顶向下"的方式。我们从本章开始思考感知的原因，基于人类的视觉系统，讨论虚拟环境到底为什么能有效地为我们服务。下一章中我

们将进入一个最高最抽象的层次——数学，给出计算机图形学所必需的一些数学基础。在第3章中我们仍将在一个较高的抽象层次，考虑光照问题。也就是对包含图形对象的环境进行光照的问题，说明3D图形渲染决定于描述环境中任一点光能量的方程的解。在其后的一章（第4章）中我们考虑人类视觉系统通过对颜色的感知来反应光照的效果。

我们然后从渲染场景的一个特别方法开始，展开对一种被称为“光线跟踪”的光照渲染方法的分析。“光线跟踪”方法是一种相对来讲比较实用的渲染方法，它模拟了环境中光照的一些全局特性。在这之后各章逐渐放松对光线跟踪的假设，说明如何以简化光照的代价构造一个能够实时渲染的系统。

然后我们转向介绍一种被称为“辐射度”的光照真实感生成方法，接着返回到光线跟踪方法的讨论，研究如何使光线跟踪取得更快的渲染速度。光线跟踪的讨论提出了很多最基本的图形操作，例如在一个显示设备上画一个2D直线。

2 这种叙述方式是一种“从后向前”的叙述方式，不同于先前的一些计算机图形学书籍的叙述方式。先前那些讲述计算机图形学的书籍往往是先从讲解画线和画多边形等底层基本操作开始，逐渐搭建二维图形架构，最后完成三维图形架构的建立。

然而，这本书的理念是要从最高层次开始，在读者需要时介绍概念和方法。所以2D直线和多边形渲染与裁剪这样的基本操作就被放在最后讲。

本书这样处理完全是基于一种认识，那就是整个世界似乎都可以看成是用平面多边形或三角形构成的。我们放松这个假设，分别在第18章讲解如何生成曲线对象的实体模型，在第19章中讲解曲面的问题。

这样我们就完成了本书的第一个部分——从某种真实到支持实时渲染的图形系统的技术路线。完成这样的渲染系统的构造过程之后，接下来在第20章我们讨论虚拟环境中的动态特性问题，在第21章讨论虚拟环境中的交互问题。

最后我们再一次简要地回顾了从真实到实时的整个过程。我们开辟了一章（第22章）综述具有照片真实感的图像的更复杂的渲染方法，另外还专门拿出一章（第23章）综述提供实时性能的一些最新技术。

本章的余下部分将讨论计算机图形学和虚拟环境的建模方面的问题。我们将介绍一些基本思想和一些基本术语，接着还要探讨一下为什么虚拟现实会对我们的工作和生活产生重要影响。

1.3　建模和虚拟环境

对象

人们普遍认为计算机图形学起源于Ivan Sutherland 发表的那篇论文，该篇论文描述了一个称为 Sketchpad的画板系统（Sutherland，1963）。Sutherland的系统（如图1-1）允许使用者使用一种称为“光笔”的指点设备交互地在一个屏幕上画图。很多我们今天可能很不以为然的概念都是出自这个系统的发明：比如“橡皮筋”的思想——即当用户在显示器上画一条线时，我们把所画的轨迹看成是一个橡皮筋，当固定终点时即在起点和终点位置上画出一条直线。还有二维观察背后的思想，包括“窗口”、“视口”和“裁剪”等概念也都是源于该系统。最重要的一点是，Sutherland创立了图形对象的概念，即每个实体都有其自身的语义和交互行

为，这一点将计算机图形学从单纯的图片表现媒介中分离出来，虽然这还只是初始形态，但不管怎样这是一门新学科的开端。

暂时我们可以将对象当做一个封装的、自包含的“事物”——具有结构和行为——典型地在软件中具体表现为一个包含数据结构和一组函数的实例。一个简单的例子是在某一空间中标记一个点。对象有一个视觉表现（或许具有多个视觉表现）和响应“消息”的行为，这构成它自己的语义（举例来说，移动一个标记到另外的一个点）。另外的一个简单例子是交互式程序的“按钮”。按钮作为软件小构件，它有视觉表现和对某些事件的响应能力（即行为），这些事件典型地如鼠标事件。 3



图1-1 Ivan Sutherland在画板系统上工作

一旦我们有能力定义一组独立但彼此可以交互的对象，我们就有了定义一个新的“世界”的可能性——遵从运动定律和具有特定行为的一组实体由计算机程序构造并组织成一个动态的世界。在应用中这个虚幻世界被这些基本实体之间的交互所维护，人们在其中能感知各种事物的存在，感知到各种活生生的图形对象，同时也能感知到这个世界中反映人们彼此交互的内在高层认知模型在发生作用。实体的行为通过自己的存在表现出来——典型地每个人由一个虚拟人来映射，也称之为*化身*，它的动态特性在一定程度上与它所映射的实际人的活动、行为和运动相一致。计算机图形学为我们提供了以无穷多样性表现形式生动地描述这个世界的能力。

这样的世界普遍被称作“虚拟环境”（VE）。Ellis(1991) 已经提出了讨论这种虚拟环境的有用的分析框架。他认为虚拟环境有三个主要的组成部分，即内容、几何以及动态特性。内容由“对象”所组成，这些对象构成了整个环境；几何包括维度、度量和环境的范围或边界；动态特性由对象之间的交互规则所组成。这些规则例如一个按钮是如何响应一个鼠标事件的，或在物理仿真中对象如何对与其他对象发生的碰撞作出响应。在下一个小节中我们讨 4

论虚拟环境的内容，在第2章中再讨论虚拟环境的几何。

VE 的内容

环境的内容与环境是如何建模有关。环境包含一组对象，如在上面讨论的。在任何时刻，每个对象都有一个*描述*和一个*状态*。描述包括关于它的几何实体的信息、可能具有的行为。几何描述一般是与具体的坐标系相关的（我们将会在第 2 章对此做更详细的讨论）。它确定在某个时间点上对象的位置和方向，以及它与其他对象之间的关系。状态确定对象的状态——这取决于对象的语义构造以及该对象与环境其他成分之间的关系。

对象的一个特别子集叫做参与者。*参与者*是一个能发起和另外的对象或参与者交互的对象。这是依赖对象间的信息交换来实现的。有一个特别的参与者，那就是操作虚拟环境的人。这个虚拟环境的参与者在环境里面有一个视觉表现（举例来说，在最简单的图形应用程序中的光标，在比较复杂的应用程序中的一个完整的具有人的特征的图形表现，或者一个化身）。一个较复杂的例子是由 Neal Stephenson 在他1993年的小说《雪崩》中提供的（Bantam Books，paperback）。Metaverse 是一个三维的虚拟世界，人能参加进来，有一个“化身”来具体表达该参与者——典型地，这个化身表现为人体的 3D图形表现，或是一些类似的形象。有一些对象，假设就是化身，以人的形象出现，实际上它是系统的守候进程，代表计算机系统维持 Metaverse 运行的进程。在William Gibson的小说（Neuromancer，Ace Books）中，这样的非人实体称为AI（人工智能）。

在图1-2中我们给出了共享的 VE 系统的一个示意性表示。假设计算机储存了一个数据库，其中存储了场景中所有的对象。每个对象有几何描述，例如构成对象几何的简单形状集合。这种描述可能是具有较高的层次（如用方程定义的对象，例如球体或圆柱体），或具有较低的层次
5 （如用一组组成对象表面的三角形来表示）。每个对象分别有关于材质属性和物理属性的信息，有时还有关于光如何反射的信息（即辐射属性信息）。还可能提供其他的信息，例如对象的声音属性（如它是个声源吗？），以及对象的行为属性（如它在被刺或被探查的情形下将如何反应?）。场景数据库包含对所有对象的总描述信息和那些必须对整个场景给出的任何其他信息。

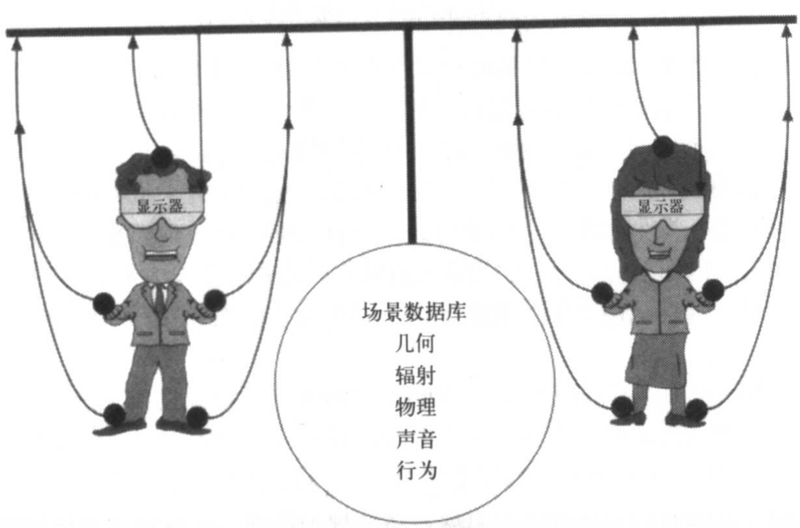


图1-2 虚拟环境系统的示意图

这个数据库一定在维持共享虚拟环境的计算机网络上的某处储存着。举例来说，它可能被拷贝到每个机器，或可能是在每台机器上拷贝了各自所需要的关键部分内容。分布和更新VE有许多不同的策略，感兴趣的读者可以参考Singhal and Zyda（1999）。网络上的每台计算机一般有一个参与者。每个参与者是显示信息的接收者，这些信息包括视觉上的、听觉上的和任何其他的信息，例如触觉或力觉反馈数据，通过传感设备从 VE 输出给人。在理想的情形下，这个虚拟传感数据提供全方位的感知体验，完全将参与者包围其间。这只是一种理想，并未能完全实现——更典型的情况是系统只包括视觉系统和听觉系统，总是有来自现实世界的信息刺激参与者，所以参与者从未完全地封闭在虚拟环境中。

图1-2 中的小圆点表示跟踪设备。参与者接收 VE 系统产生的传感数据，同时他们自己的运动也被跟踪并反馈进入系统，特别是跟踪系统会将人头部运动和身体运动信息映射到虚拟 6
环境中来。举例来说，参与者将头转向一个特定的方向时，将会看见和听到 VE中的特定的一些东西，改变朝向时，会感知到另外的一些东西。另外，参与者可以弯下身子，可以从虚拟桌子的下面看过去。这些跟踪数据能确定参与者的头和身体在哪里，因此系统能根据人的视点和身体朝向恰当地调整要显示的知觉信息。一般情况，在一个 VE 系统中只有头部和一只手的位置和方向被跟踪（见第 20 章和第 21章）。

在一个VE 中当参与者低头看他或她自己身体的时候会发生什么情况呢？有两种情形有可能发生。第一种情形，也是最通常的一种情形是，参与者穿戴一个头盔显示器（在下面所述），该显示器显示的虚拟环境随着头部的移动而变化。场景数据库中对象之一是参与者人体的表示，该虚拟人体将会依照人的行动而改变：当他伸出手的时候，对应的虚拟手将会伸出。因此，这个化身的场景数据库描述是依照化身所表现的那个人的跟踪信息而更新的。在共享的VE 中，如果位于另外的机器节点上的一个参与者观察一个移动中的人，他将会看见表示那个人的化身在移动。对场景数据库的更新一定要以某种方式及时反映到分布环境的所有节点上，尽管会有网络延迟现象存在，在共享虚拟环境中的人们能体验相同的（虚拟）真实。

也存在另外的一种可能性：参与者可能是在一种称为“CAVE”的虚拟环境中（见彩图1-3和图20-2）。“CAVE”是 CAVE 自动虚拟环境的缩写（Cruz-Neira et al.，1993）。CAVE 这个概念和它的第一次实现是 1992 年在芝加哥伊利诺州立大学。理想的“CAVE”是一个拥有六面墙壁当做投影屏幕的房间，虚拟环境（VE）被投影在这些屏幕上面。一个参与者穿戴轻型的立体眼镜和一个头部跟踪设备。显示器上交替地显示出左眼图像和右眼图像，立体眼镜能保持与这些显示同步，只有左眼的图像允许进入左边的眼睛，同样地，只有右眼的图像能够进入右眼（参见下面的有关立体的讨论）。头部跟踪器是用来计算穿戴者的眼睛位置的设备，基于两眼间距，为CAVE六面墙中每一面计算出立体投影。立体眼镜采用的是一种快门技术，使用该技术能将左眼和右眼的图像帧交替地呈现给观察者，这样该观察者就完全沉浸在周围的 VE 中了。实际的“CAVE”一般采用四面墙壁作为投影屏幕，分别是前面、左面和右面以及地板作为墙壁显示器。投影系统和软件使得参与者一般情况下不会注意到物理房间的拐角。多人可以同时处于“CAVE”中，虽然显示内容的更新完全地只针对其中的某一个参与者。“CAVE”也能使多个身居遥远位置的参与者共享，代表不同物理位置处人的化身将共存于“CAVE”中，这样分布在不同物理地点的人们就能同时存在于同一个共享空间中了。不像头盔显示器，在“CAVE”中的人将会看见他或她自己真正的身体，如图1-3所示（彩图）。然而，在遥远位置的人看见的是表现他人的化身。

1.4　真实感和实时

虚拟环境必须被显示出来，同时参与者必须得到实时跟踪，这样虚拟环境才能有效地工
7 作。另外，所显示的内容一定要充分真实，以使人们可以产生并维持对相应现实的幻觉。

然而，在这两个需求之间是存在冲突的。在讨论它之前，让我们站在计算机图形学的角度考虑术语“真实感”的多种不同含义。

几何真实感

所谓的“几何真实感”，我们指的是一个与要描述的真实世界中对象具有十分相似几何外观的一个图形对象。举例来说，一栋建筑物的一个完全而详细的建筑设计图，不管该建筑物是否真实地存在。虽然这样的设计图通常都是二维工程图，我们可以从完全的 2D 信息构造出该建筑物的3D表示。这样，每面墙壁、窗口、空间分割、房间中的每个内部对象（如灯、桌子、椅子等）都会如它们的真正实物一样，具有完全相同的尺寸。

典型地，3D 几何表示是用平面多边形（一般是三角形）来构造的，由这些平面多边形拼
8 接成对象的表面。对象表面某些区域越是弯曲，所需要的三角面片数目就会越多。只有这样才能保证虚拟现实和物理现实之间充分相似和一致。

图1-4 给出了一个人脸图像，这是从UCL[⊖]开发的光学表面扫描仪上获得的（如图1-5）。

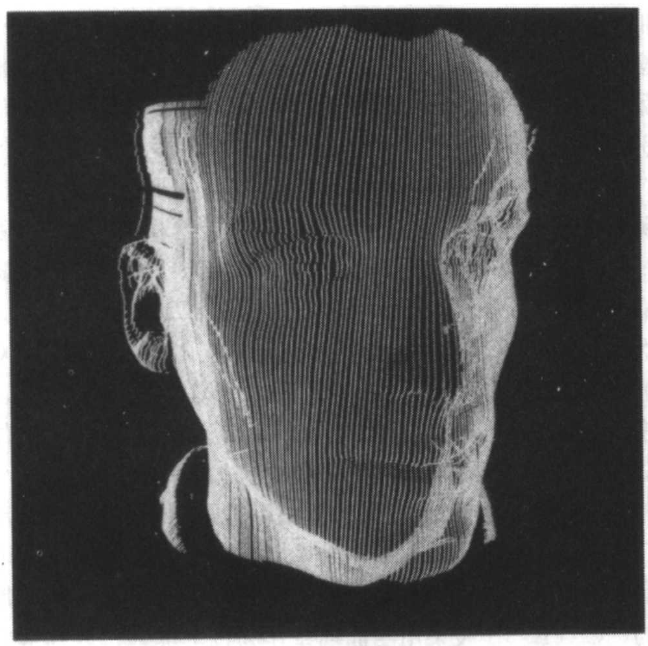


图1-4　人脸的激光扫描产生的3D 点云（由UCL计算机科学系的Joao Oliveira提供）

所显示的这张脸由64 028个(x，y，z)点（也称作**顶点**）所组成。该扫描过程大约进行了10 秒。为了要让这些点可使用于 3D 图形显示，它们被组织成三角形的形式。在这种情况下，我们
9 获得了126 108个三角形。表面可以进行明暗处理，生成的图像如图1-6所示。这可以在三维空间中旋转，这样就可从不同的方位显示它。

⊖　http://www.gene.ucl.ac.uk/face/。

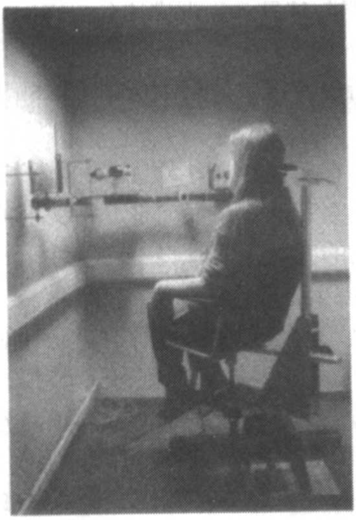


图1-5 人脸扫描仪（由UCL医学系的Alf Linney提供）

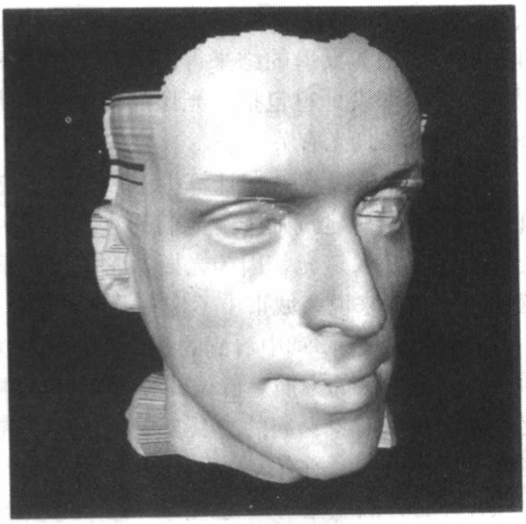


图1-6 对激光扫描的人脸的渲染（由UCL计算机科学系的Joao Oliveira提供）

几何真实感的一个实际用处在于虚拟原型设计。对象、装置、飞机、建筑物或其他物体都需要在建立真正的物理实体之前有一个虚拟的构造过程。这可能是设计过程的一部分，或者是检查对象安放在特定位置上所产生的影响，抑或在现场试验设备的可用性。

彩图1-7给出的是“伦敦千禧眼”的一个虚拟模型。这个模型的构造是作为对空间使用的设计过程的一个部分。注意到模型的几何构造是非常精确的，但看起来仍然显得不很真实，这是因为明暗处理的光照效果不够真实。

彩图1-8给出了一个非常有趣的虚拟原型设计的应用。Maitreya Project公司打算在印度Bodhgaya建造一尊500ft（152.4m）的佛像——Maitreya Buddha大佛。这个佛像将存在1 000年，建在一个风景优美的公园中，该公园中有寺院、学校、医院和研究中心。图1-8a给出了一个由艺术家做的该佛像的物理比例模型，通过对它进行激光扫描来生成其虚拟原型。产生的虚拟模型可以用来评估位于Bodhgaya的这尊巨大雕塑（见图1-8b）。通过在虚拟现实中观察模型，人们可以实际地穿过公园，了解这尊真实雕塑落户实际公园中会是什么样子，甚至想像出在最终落成很多年之后会是什么样子。虚拟模型也可以被艺术家用来改进对佛像本身的设计，比如通过虚拟模型试试各种不同的面部构造。佛像位于宝座之上，宝座也已经做成了虚拟模型。

虚拟原型设计应用创建几何真实表现有一个基本原因：几何通常被用作待建的真正物理对象的蓝图。达到几何精确性的另外一个理由是虚拟的模型可以被用来对实际的安装过程进行预演。举例来说，消防泵模型可以被维护工程师用来练习和进行维护辅导（见彩图1-9）。如果几何尺寸不够精确的话，那么这种虚拟培训就可能带来负面作用——受培训者使用在虚拟世界中学习的技术于真实世界中时，会使得结果变得更坏。

光照真实感

在图1-8b中的虚拟佛像看起来很真实，这不仅是在几何意义上，而且也在光照方面很真

实（虽然在这一点上周围环境并不是这样）。事实上光照的计算是根据佛像在年内一个特定的日子里对太阳光反射的结果得出的，考虑到那天中的特定时刻太阳光投射的角度。在这个应用中虚拟原型设计也包括对光照的考虑——大佛在场所中的视觉效果不仅简单依赖于周围的
10 环境，而且与日光有关系。

这样的光照真实感在计算机图形学的许多应用中具有重要意义，最显著的例子就是在建筑设计中。它对建筑师了解他们的建筑在不同光照条件之下的效果如何是非常关键的，无论是自然光还是人造光，因为光照通常不仅仅具有美学上的意义，同时也在功能性上具有重要意义——举例来说，城市中不应有照明死角，因为那会滋生犯罪。

如我们将在第3章中看到的那样，正确的光照计算是一种极端复杂的计算，是计算机图形学中计算最精深的方面。佛像的光照相对来讲还是简单的——因为只有惟一一个对象被照亮，此时根本没有对象之间的交叉反射现象需要考虑。然而，一建筑物场景内部的正确光照却一定要考虑到场景中所有表面之间的交叉反射，而不能只单独考虑每个对象上的光照效果。

后者的方法叫做*局部光照*——每个对象上的光照好像它是场景中除了光源之外惟一的对象。*全局光照*指的是对环境中光分布的正确计算，对象与对象间的反射要被考虑进去。由这种方法生成的图像有时被称为*相片逼真的*，它意味着大体上这样的图像应该与用真正的照相机拍摄出来的那些在适当的真正光照条件下的景物没有差别（彩图1-10）。

行为真实感

一个图形对象可能根本不是任何真正的东西的几何表现，或者它也许是对某种真正的事物所做的一个巨大简化，而且它的光照可能与真实世界中的完全不同。尽管如此，这样的对象对于观察者来说在某些意义上是完全“真实”的：举例来说，虽然是一个人的粗糙描述，但不管怎样，他都能引起观察者的情感共鸣。

彩图1-11给出了一个来自某个虚拟现实应用的一个快照。该应用与开发治疗那些有当众讲话恐怖症的病人的治疗程序有关（Pertaub et al.，2001）。通过这个图像我们能清楚地看到，
11 所描述的人物并不真实，而且会议室同样不够真实。人们走进一个虚拟现实并在一群虚拟听众面前说话。当听众有怀敌意或无趣的方式举止时，相比听众表现出感兴趣和友好时，讲话者表现出更高层次的焦虑。然而，讲话者确实知道在那里实际上并没有“听众”。这种反应对于虚拟人物来说不会由几何真实感而引起，因为很清楚，这些人物并不是几何真实的。反应是由行为真实感的程度引起的——听众成员有动态的面部表情、眨眼、用眼神与讲话者交流、在座位上坐立不安，并且通常会做出一些夸张性的动作——对听众的友好的或带有敌意的刻画。它说明演讲者在虚拟听众前禁不住要对这些行为响应，即使他们完全清楚这样的听众行为完全是计算机仿真程序预先编制好的结果。

通常卡通能非常奏效是因为它们具有行为真实感。请想像一个著名的卡通形象，例如唐老鸭，它看起来根本就不像。然而我们能认出它而且对这个角色的情感表达产生共鸣，事实上我们都非常快乐地享受着它的冒险历程。这些故事一直家喻户晓，就好像电影中所描述的是真实事件似的。制作生动卡通形象的艺术在很多卡通创作公司如迪斯尼得到深入研究，卡通艺术对我们理解在情感刻画方面什么才是重要的具有重要价值（Thomas and Johnston，1981）。

漫画、印象主义和图标表现

图1-12是一个名人的一幅漫画。我们全都能立刻认出这是美国前总统克林顿。从真实的

角度来看，它根本就不像那个“真正的”总统。我们能认出这个总统是因为卡通依赖于存在于我们脑海中的对该总统的内部视觉认知模型。它夸张了某些主要特征来匹配我们脑海中的内部表示。卡通所包含的大部分内容事实上是由感知者思维活动完成的，映像“存在于观察者的脑海中”，这是一个智者名言。了解哪些是本质特征并夸大它，以便呈现一 12
个有效的幻影，这是艺术家的技术。

在艺术世界中总是存在着产生不同表现手法的潮流。各种不同的现实主义学派相信艺术的目的是表现“现实”。

现实主义本质上导致一种静态的、视觉的简单印象，反映的是一个冻结了的时刻。印象主义则放弃对视觉精确的追求，转为提倡对运动和活力的反映，这样的绘画能刺激许多感觉，不只是视觉感觉。我们审视梵高的油画（见彩图1-13），这是一幅夜晚河景的图画，它显然不是视觉上的真实。然而它给了我们一种很强烈的运动印象：微微泛光的河水、摇动的小船、一对老夫妇漫步在水上，似乎能听到潺潺的水声。这幅画描绘了一个活生生的场景，显然它不是相片逼真的，但很容易引起置身在那里的想像。

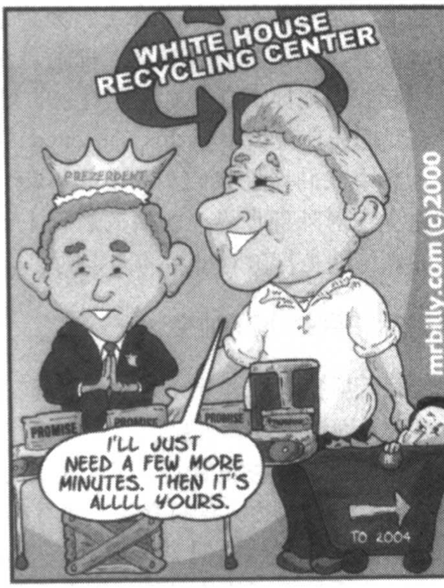


图1-12　美国前总统克林顿的漫画（由Billy O'Keefe 提供）

图1-14给出了一个孩子笔下的人物。请看它——
当然它不像任何一个人，但我们仍然能立刻明白她/ 13
他想画的东西。这里的不经意的技术被称作图标表现方法。以某种方式，通过内在的对“人”的视觉认知模型的模拟，将图画转换成了对人的描述（什么是一个人?——包括一个头、一个躯体、二只手和二条腿）。这种图标表现是如此的有力，以致于我们绝大多数人实际上受到它的阻碍，无法成为绘画能手。当我们画一个对象的时候，即使努力去研究它并试图画出所看见的，我们还是容易将它画成我们脑海中的印象而不是实际所看到的东西。换句话说，我们看着一个对象，但是我们并没有看见它。我们画的是我们的内在模型，而非基于从外部世界来的感觉信息在作画。



图1-14　一幅孩子的画，由六岁的Amy Goldstein提供

Betty Edwards（1999）在她的一本很著名的关于绘画艺术的书中，说明了我们如何能克服干扰我们的图标表现而变成优秀的绘画艺术家。举例来说，试着重新去画上面那幅克林顿总统漫画。在第一次尝试时，除非你已经擅长画画，结果将不会太好。现在将这幅画翻过来，并照着它再描一遍，专注于你所看见的笔划，尤其现在正在画的笔划，因为此时所看到的原画是反的，因而它失去了唤起我们内在认知模型的能力。同样道理，当我们在现实生活中取材作画时，努力将注意力集中在表面之间的空间中，因为它本身是没有什么意义的，而不要

将注意力放在那些表面本身上，因为它们是有意义的。该如何去看是要经过一番训练的，这种训练要压倒那种心理倾向，即通过观察然后让感知信息触发我们内在的表示，这种内在表示引起我们去搜索自己对那个对象的图标表现。

我们先前曾引用Carlos Casteneda的话开始本章的内容。它是“魔术师眼中的世界”，在这个世界中我们看见我们头脑中的事物，我们看见我们所预期看见的东西。我们的大多数生活都建立在对外部世界中物质的解释上，经过了感知系统、文化系统和观念系统的过滤。魔术师的训练就是打破对世界的固定的图标解释，以一种新的方式来观察这个世界。

真实感和实时之间的冲突

对于虚拟现实程序来说，人类通过图标表现方法认识世界的趋向的确是非常幸运的。为什么呢？因为在计算机显示器上产生“现实”是极端困难的。虚拟环境依赖于我们自身的能力，即从现实的极小样本概括形成我们对整个现实的经验的能力。我们看着一些看起来像“头”、“躯体”、“两条腿”和“两只手”的线条时就能看见一个人！要想看到图1-15平面里的各条线段几乎是不可能的，而且不会看见3D立方体（事实上是两个交替的立方体）。计算机图形学通常目标是真实，但是事实上结果却常是高度印象主义的，尤其在动画中。在图1-11中对听众的描述就是非常印象主义的，它和卡通一样都离视觉真实感有相当的距离。然而这个应用所产生的焦急和情绪反应的范围和程度与患者在面对现实生活中的听众时是相同的。

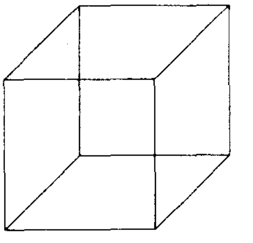


图1-15　Necker立方体

14

计算机图形学中理想的“真实感”包括上面所有的各项内容：几何视觉真实感、行为真实感和光照真实感——它们的难度是依次递增的。在计算机图形学中的基本冲突就是真实感与实时，上述的这些真实感只有在牺牲实时性能的前提下才可以达到。然而实时性能是保证虚拟现实充分发挥作用的必要条件。

在这样的上下文中，“实时”有多种含义。首先我们注意到计算机在显示器上产生运动和变化幻觉的方式与卡通或电影是完全相同的。计算机在每秒钟内以正确的顺序产生和显示许多幅图像。每一幅图像叫做一个帧。帧的频率指的是每秒实际显示出来的图像数目。这通常是以Hz为单位的。30 Hz的帧频表示每秒显示出30幅图像。如果帧频足够快，那么观察者将会看到连续变化的场景而非单独的个别帧图像。换句话说，很多单个帧图像以时间顺序串连在一起，产生出了一个可信的连续体验。与卡通和电影相比，卡通和电影总是以相同顺序播放一帧帧图像，一旦动画开始，就不再有人的干预。然而，在虚拟环境中，显示的东西依赖人的干预，因此没有固定的图像顺序，下一帧图像是否显示决定于虚拟环境中的人要做什么。这有很多种情况可能发生。

实时漫游。人作为虚拟环境的参与者可以在虚拟环境中走动，并且可以随意环顾周围，在意图和结果之间没有引人注意的延迟存在。举例来说，参与者形成将他或她的头转向右侧的意图。将这个意图转换成转头这个物理行动大约有100 ms的时间延迟。我们假设头的转动已经被跟踪而且传送到计算机，计算机必须连续且平滑地更新显示，以便显示出与头的方向相应的图像视图。如果一个人站在一栋虚拟建筑物之前（彩图1-16），当他将头转向右侧时，一部分建筑物似乎在移向左侧，而不断有新的部分从右侧进入视野。在计算和渲染下一帧图像时将会有一个时间延迟，或显示滞后。这种滞后实际上是随时间改变的——举例来说，当

一个人俯视街道的时候，应该有非常多的房子进入眼帘，因此有较多的渲染需要做，这肯定会花掉比较长的时间。相比之下，当参与者直视一面几乎遮挡了所有景致的砖墙的时候，相当少的渲染工作需要做。当然，在全部滞后中还包括另外的一个成分，这就是由跟踪设备将转头信息传递给计算机程序的时间耗费和处理器处理相应数据的时间耗费。理想情况下帧频应该在每秒60帧左右。在每秒10帧情况下，世界将会显得非常不平稳，而且沉浸在这样的一 15
个世界中可能导致仿真者呕吐，就像晕船和晕车一样。一般电影的播放速度是每秒24帧图像，现在绝大多数工作站的显示帧频都至少有60Hz。

实时对象交互。在这种情况下，参与者有办法通过对对象操作来改变环境——举例来说，拾取某物并摆放它在另一处（彩图1-17）。典型情形下，参与者至少有一只手被跟踪，并且可能有一只虚拟的手在 VE 中与之对应。计算机程序在整个过程中都有关于虚拟手在 VE 中位置的记录。当手与某一个对象相交时，一种适当的触发机制被激活，然后手就能抓取对象，并以各种不同的方式移动它。就计算机图形渲染而言，这与漫游没有什么不同，每个帧要渲染的场景依照的是其中每个对象的当前状态。当虚拟手移动的时候，它在虚拟世界中的位置在场景数据库中得到更新，所以在下个渲染时刻，它就会在新位置上出现。当然，如果有滞后存在，参与者可能会在移动他们手的时候发现实际上虚拟手的移动要慢得多（超过 100 ms 会引起注意）。这个滞后会造成在虚拟环境中选择和操作对象的过程变得非常困难。

在共享的虚拟环境中交互。上述的例子是假设应用程序运行于一台计算机上，有与它关联的显示器和跟踪设备，服务于一个参与者。然而，由位于全世界不同地理位置的多个参与者共享的虚拟环境并不是罕见的（Steed et al.，1999）。每个人当然有一个单独的机器和显示器，但是描述在VE里面发生的事件的程序和数据却是分布在网络上的（例如因特网）。举例来说，参与者A在伦敦拾取并移动一个虚拟对象。数据表明所发生的一切都通过网络传送到所有的其他参与者，他们的局部场景数据库的拷贝一定要得以更新，他们的显示也要发生变化以反映这些改变。想像参与者A尝试与远在数千公里之外（例如在旧金山）的参与者B“握手”或者将一个对象传递给他。虽然A拾取了对象，但是可能经过数秒之后B才能看见该对象被拾取，所以A看不到B的反应，因此会决定再一次放下对象；与此同时，B正在接近该对象……很显然，这引起了系统另外的一种延迟。无论在共享虚拟环境中的每个局部位置上的运算速度有多快（延迟有多小），网络上的数据传输所需要的延迟是系统所无法避免的。每个位置可能以每秒60帧的速度运行（60Hz），但是即使一个很小的延迟如500ms，也会完全破坏参与者之间的正常交互。（Ellis和他的同事们对延迟问题做了广泛的研究，参见Ellis et al.， 16
1999；Ellis et al.，2000。）

因此，实时需求对计算、设备、显示和网络性能等方面都提出了极高的要求。交互者的意图应该被计算机“即时”地了解。事实上，如同我们将在下一小节中所要看到的，即使是在真实的世界中，交互者的意图也没有被即时地了解——在这里人类的知觉和生理系统本身存在着一个延迟。所以，假如虚拟的事件发生在人们的响应时间允许的范围之内，事件就认为是即时的，但是即使是这一点今天的系统也远未达到。

因此采取折衷是不可避免的：今天即使是在实际光照效果的虚拟环境中做实时漫游也是不可能的——光照真实感的一些方面必须被牺牲掉。在一个复杂场景中，通常完整地显示每个对象的几何真实感也是不可能的。精确地表现一个人体可能需要数万个多边形——想像如果有一个应用，在那里有数以千计的人（例如虚拟的人群），情况会是怎么样？在今天的硬

件上实现这样的实时应用是不可能，所以人们提出了各种不同的简化方法。在极端情形下，一个人体可以由一个多边形来表现，在该多边形上描绘一个人体的图像（参见Tecchia and Chrysanthou，2000）。显示对象的所有姿态和运动的特性对计算能力来讲，比全局光照下对象渲染具有更高的要求。一个好的例子就是布料（彩图1-18）。有关布料的物理方程是十分复杂的，在这种情况下，用好的物理模型对布料做实时动画也是不可能的。同样，这里需要做一些折衷，采用对物理定律做某些简化的处理方法。

最后，在网络支持的共享虚拟环境的情况下，总是存在着各种各样影响很大的网络延迟。要求在任何时刻任何局部节点上都能确定所有对象正确的状态是不太合理的。通常要采用预报算法，将网络状态考虑在内，并对虚拟世界的状态给出最佳估计。

正如我们所说过的，在真实感需要和实时需要之间存在着固有的矛盾。计算机图形学及其在虚拟环境中的应用就是一个折衷，它的效果是非常好的，这得益于人类奇特的视觉和知觉系统特性。这种对现实的低频采样是必须的，这是为了维持一个虚幻的现实。我们在下一小节中更详细地讨论这个内容。

1.5 存在和沉浸感

在1999年的夏天，UCL（伦敦大学学院）做了有关在虚拟环境中存在概念的系列实验中的
17 一个。存在是通常不被注意的一种感知，它是有关在特定场合并对其中事件做出反应的一种描述。这种未被注意的现象是因为对于我们绝大部分具有清醒意识的生命来说，关于我们在哪我们从未怀疑过，这里没有任何秘密。我们在我们所在的地方。

但是我们在哪里？

人类可能具有一种独特的想像能力——从进化的角度来看，想像使人类能够设想将要发生的事，为自身的生存规划各种不同的策略，很显然这是一种自然选择的结果。但是想像也包含对那些从未有过也不会发生的一些情节的构想，而且我们能以某种方式在这些情节中找到自己。当我们看一本小说的时候，虽然不是“在”小说中，但是在强烈的存在描写面前我们仍然对叙述做出实际反应。举例来说，当我们看到一段特别戏剧性的情节时，我们的心会怦怦直跳，而当看到一段十分悲惨的段落时泪水又会禁不住夺眶而出——即使我们完全清楚事实上根本没有任何事件发生，或者说，这些事件完全存在于小说中，我们只是在做看小说这样的行为。我们能对故事产生反应，但是如果在小说中的人物发出尖叫声——“着火了！”，我们根本不会担心有生命威胁，不会跑到街道上。最多我们可能让自己想像逃避火场情景，或是让小说的人物逃离火情，我们会不安地将小说翻到后面去看这一戏剧性“事件”的结果。

你置身于电影院中。事实上你正坐在许多人中间，或许该电影院离家不远。不一会儿，你乘上了一艘星际飞船在遥远的未来时空中翱翔。偶尔能听见电影院里有人咳嗽，或是听到周围有人摆弄爆米花袋子的声音，或许你会觉得太冷或是太热，或许还会有一些奇怪的声音发出来，使你立刻意识到此时是坐在电影院里，坐在很多一样在看电影的观众中间。但是这种“回到”电影院的意识本身就意味着在某种程度上你已经暂时离开了这里，到了别的什么地方。电影中的事件会让我们心跳加速，让我们从位子上跳起来，或是由于恐惧而将视线移向别处，或者眼泪自然而然地落下来。但是同样，如果搭乘的星际飞船着火了，你不会冲出电影院。你会尽全力让主人公逃离危险，但是这仍然不是实际活动。一方面，你产生了在真实物理环境之外的存在的感觉，但是从你自身所处的位置和状态看，你确实是存在于电影院

中。电影调动了你的感觉，但是身体的反应是受限制的——一些不自觉的反应如心跳加速、出汗、自然而然跳起来或是感到透不过气等。但是你和电影院里周围的观众一样，还是稳稳地坐在座位上。

有关电影经验的另一个有趣特征是，虽然在某种意义上是一种共享的经验，很多人在同一时间经历同样的事件，但这并不是说所有的观众都与你一样沉浸在电影当中。即使同最要好的朋友一起去看电影，当看的是一部非常好的电影时，你可能忘记了这些朋友的存在。直到电影结束，你们一起走出来并交换着感受，评论电影的好坏。

现在推想它是一部3D电影，你戴上一副特殊的眼镜，这种眼镜整合左右眼视图，给人一种立体的视觉感受。一颗导弹迎面飞来，直达我们的面部——这时你很有可能低下你的头以免被击中。尽管你心里知道并没有真正的导弹，但你大脑中还是有一部分不知道它是虚拟的，从而做出真实的反应。当3D电影中某人呼喊“着火了！”的时候，你还是不太可能从你的位子上跳起来冲出电影院。事实上，只有当你坐在位子上的时候，你才停留在电影的世界中。离开位子就意味着你与电影导演之间的不言而喻的契约的终止。只有当你呆在位子上的时候，你才存在于电影的世界中，但是呆在位子上也同时是不存在于电影情节中的一个信号。你的注意力在电影情节和现实的电影院之间不断转换，尽管对于电影情节有一些身体上的反应行为，但是你的总体状态还是在电影院中。

假设你正在家中电脑上玩一个改编自那部电影的游戏。你完全地投入到游戏当中去，这是一个在线游戏，由世界各处数以千计的人们所共享的游戏。同样，如果游戏中的一个角色大喊“着火了！”，你的心跳在加速，然后操纵你的化身逃离危险。当然，你有许多身体反应，当逃逸仓受到来自主船的爆炸冲击时，也许你在座位上摇来摇去，但是你的身体必须保持在现实中的椅子上，以使你的注意力停留在那个虚拟的世界中。这里也存在一些交叉，也就是说，虚拟世界中的事件在有限的范围内也会与真实世界发生交错，导致你在真实世界中的运动和行为，但是这些运动与你在真实情景中的行为是完全不同的。 18

假设你在床上；这是一个特别强大的电影和游戏，一同构筑起一个梦，一个难以忘怀的真实的梦。你完全沉浸在电影的情节中，在星际飞船上，雇佣战神进行战斗。现在有人呼喊：“着火了！”你当然会竭尽全力逃离现场……只是你没有动，你正在床上熟睡。在梦中，运动神经的活动已经停止，身体基本上处于麻痹状态。你的自治神经系统当然是在工作，使你保持着呼吸，一些感觉神经仍然是警觉的，帮助你避免某些危险（例如烧焦的气味将会唤醒你），但是你不能移动你的四肢。在梦中，你的眼睛快速地从一侧移动到另一侧，即所谓的REM（rapidly eye movement）睡眠。对REM睡眠的发现（Aserinsky and Kleitman，1953）是在理解睡眠周期和理解做梦方面的一个重要突破。如果你在REM睡眠中被唤醒，你的梦就被打断了，但是如果你在非REM睡眠中被唤醒，就不是这样（LaBerg，1985）。所以，做梦在另一种现实中提供了很强的存在感，这时只有你的真实身体是实实在在地在你的卧室中，在真实的现实中。然而，通常梦包含两种存在的现场的重叠（梦的现场和真实现场）。Sigmund Freud（Freud，1983）报告了结合发生的实例——即在做梦人物理世界中的一个事件变成了梦境中的一部分。Maury有关法国革命的那个著名的梦是这样记载的：在他熟睡时一张纸落在他的颈项上，梦中的现象被解释为断头台上铡刀的落下。 19

在梦中会有希奇古怪的事情发生（图1-19），但是我们做梦的大脑没有询问这些事件。让我们回到Carlos Casteneda的魔术师Don Juan，一个有能力学会在做梦的时候保持清醒的人。要达到这一点需要在日常生活中保持判断力：我在哪里？这正在发生吗？我是如何到达这里

的？这种比较警觉的态度能扩散到我们的梦中，通过这样的反省我们能变成做梦时保持清醒意识的人。Don Juan 提议建立一条指令去看你的手，然后进入梦境，然后再一次看你的手，如此方式帮助稳定梦的情节。现在在这个梦的状态中，你完全能意识到它是一场梦，你就会意识到你的梦的表面物质，而且能通过练习，任意改变梦情节——就好像你已经变成了电影导演并在影片中扮演角色。你能用你的所有由梦中意图所产生的行动来熄灭大火。你在梦的情节中清醒地存在，但是你的物理存在还是在你睡觉的地方。斯坦福大学梦研究实验室的Stephen LaBerg 已经对这样清醒的梦做了广泛的研究（LaBerg，1985）。他利用REM睡眠的眼动，清醒的做梦人通过它可以向试验者发出信号告诉梦的开始（同时停留在梦中），这样第一次在做梦的人和外部观察者之间打通了一个简单的通信通道。



图1-19 关于预言家Jacob的梦的一个雕刻

在位于小礼拜堂山的北卡罗来纳州大学的图形和图像实验室有一个大范围的跟踪系统，安装在宽4.5公尺、长8.5公尺的整块天花板上，有超过2米的高度变化，每秒2 000个样本，误差不超过 0.5 毫米（Welch
20 et al.，2001）。它跟踪VE中参与者的头部位置和运动，参与者穿戴一个头盔显示器（HMD）。HMD分别传送左眼和右眼图像，最后形成一个完整的 3D 立体视图。每当参与者转动他或她的头部，他们仍然将会从VE中看到完整的视觉流——真实的世界被完全封闭在外面。HMD也能传递周围环境的声音。因为头部在一个广泛的区域内得到跟踪，参与者也在附近走动同时完全沉浸在虚拟环境中（Usoh et al.，1999）。在某实验中，虚拟环境展现的是个厨房情景，在现实世界中那里有一些简单的石膏板复制品放在厨房中，在虚拟环境中的相应位置也有对应的对象存在。这样VE形成了在真实空间上叠加了一种虚拟。这两个空间是完全一样的，而且在实验室中物理对象和它们在虚拟环境中的对应物之间是一致的。如果一个参与者伸手去触摸厨房的桌面，他或她会在真正的世界中感觉到石膏板——这样就从 HMD 将触摸的物理感觉叠加在视觉感知上了。

现在想像你就在这个情景（虚拟的厨房）当中，你可以在其中任意走动和检视。假设你在厨房和邻接的房间走了一个来回，伸手并触摸到了某件东西。现在你又在虚拟厨房中了，突然你看见在火炉之上有某物着了起来。你闻到了燃烧的味道，感到了来自那个方向的热量。你听到某人呼喊："着火了，快出来！"你知道你是戴着头盔站在实验室中。下意识使得你想要尽快走出厨房。你知道那是荒谬的行为，事实上并没有"真的"着火，而且甚至在实验开始之前实验者就曾告诉你将要发生这些，但是不管怎样，你还是感觉从厨房出来比较安全，于是你离开了厨房并与现场保持相当一个距离。不但你的心跳在加速，而且你也确实走出了着火的地方，这是一种完全的投入。

上述的实验并没有真的去完成。然而，UNC 和 UCL 合作完成了一个比上面试验更简单的实验，同样是假设会让你从虚拟厨房走出来，结果非常令人吃惊。在这个虚拟环境中包含一个小的房间和邻接的一个较大房间。在小房间中实验主体学习该如何拾取一个方盒，然后他们被指引将方盒移进第二个房间并放在房间里侧的一把椅子上。这看起来好像是一个十分简单的任务。

彩图1-20给出了第二个房间。从该房间的外面看好像很正常，但是一旦你走进去，你发现地板的中央完全是空的，往下看，可见一个大约深度有 8 公尺的深坑。所以要走到另一边的椅子处，你要么小心翼翼地紧靠着墙边走，要么径直穿过空的区域到达椅子处。 21

你知道在那里没有深坑，如果你走入空处也不会有任何事情发生。尽管是这样，虚拟环境中的主体绝大多数选择了沿着边缘的那条难走的路径，其中很多人有非常强烈的眩晕感。实验室的来宾都反映说在深坑边缘时多少都产生了一种不安的感觉，同时作出了一些与在真实世界中极为相似的反应。他们心里很清楚不会有任何伤害，然而知觉系统的主要部分还是不了解这一点，由此产生了对所见现象的反应：那儿有一个深坑，非常危险，我一定得避开它。这种存在意识发生的现象已经被应用于恐高症患者的精神疗法（Rothbaum et al.，1995）。这种反应与图1-11 的应用中那些当众讲话人的反应是一样的。在那里并没有听众，但是从反应上看却好像有听众存在。

在沉浸式虚拟环境的情况下，相似性是非常高的。在虚拟世界中参与者全身心地对事件产生反应。我们前面所举的有关着火和深坑的例子是相当生动的，其实这种相似性在非常一般的情况下也会发生。例如，在沉浸式虚拟环境中，当我们要从桌子底下看过去的时候，参与者就必须弯下腰；要拿位于高处的一个东西时，比如放在虚拟书架上的虚拟书，参与者就需要踮起脚尖伸手尽力去够。当我们要向后看时，参与者需要将自身转动180度。参与者能上下地跳跃。因此存在的一种解释被定义为从虚拟环境到真实世界相似性的程度：参与者在虚拟世界中所做的与在现实世界中所做的相似行为的程度越大，存在的程度就越高。这也是一个反馈回路，因为存在的感觉越大，参与者使用他们的身体就会越自然。 22

我们已经使用了沉浸式虚拟环境这个术语，并且谈论了沉浸感。一些作者对沉浸感和存在这两个术语不加以区分，但我们还是倾向于区分它们。存在是一个意识状态，一种位于虚拟世界中的状态，而且正如我们前面所讨论的，存在有它的行为信号。只有当我们有至少两种可比的环境（如实验室现实环境和通过HMD或其他设备体验到的虚拟世界）时，谈论“存在”才是有意义的。设想有一处热的沙漠这样一个虚拟世界，但实验室真实世界是冰冷的，那么参与者在任一时刻应该响应哪一组信号呢？如果只有来自一个环境的信号，那么就没有存在的问题，参与者存在于那个环境中，可是我们知道这是无用的信息。

对一个环境的沉浸感同时与来自那个环境的传感数据的质和量两个方面紧密相关。它是计算机系统多方面的能力水平，包括传递周围立体声3D环境的水平，这种能力将封闭来自真实世界的感知，提供多种感知通道的水平，以及丰富的表现能力（见Slater and Wilbur，1997）。

这些是虚拟环境系统的可测度的方面。举例来说，有两个虚拟环境系统，其他的情况都相同，如果一个允许参与者转动头部从任何方向观察到VE视觉信息，而另一个虚拟环境却只允许参与者从某个固定方向观察VE视觉的信息（比如在小屏幕上看），那么我们就说第一个系统具有更高的“沉浸感”。给定两个系统，如果其中一个有比另一个更大的视域（FOV），那么第一个系统就比第二个系统具有更高的沉浸感。视域是视觉系统对着的角度范围。正常视觉大约让我们在水平方向能看到180度范围内的景物，在垂直方向上能看到120度范围内的景物。典型的 HMD 只能在这两个方向上提供大约60度和40度的视域（FOV）。另外一个例子，如果一个系统能实时产生阴影而另一个则不能，那么同样，第一个系统具有更高的沉浸感。这是显示方面的例子。最后，如果两个系统在除声响方面之外完全相同，一个能提供声音而另一个不提供声音，则前者更具有沉浸感。这些是有关“沉浸感”多或少的例子。显然，我们可以对系统的这些属性建立一个矩阵，这优于对参与者反应的度量。这是因为存在是一个

人的反应，而沉浸感是对系统自身的描述。

上面所描述的沉浸感只是沉浸感的一个部分：都是有关对传感数据显示的，这些传感数据包括视觉、听觉和触觉，所谓触觉指的是接触的感知和力反馈。严格地讲，沉浸感还应该包括对热的感觉、嗅觉（味道），虽然这些在 VE 中很少得到处理。沉浸感另一个部分是跟踪的范围和功效。典型的VE系统跟踪参与者的头部和一只手。跟踪系统不断地对头部和手的绝对位置和方向采样，然后这些信息被送入计算机，计算机根据这些信息更新数据库中对象的状态，并执行相应的行动响应碰撞事件。至少在参与者的身体移动时，系统中所对应的化身表现能同时改变。当被跟踪的身体部分和其他一些对象发生碰撞时，系统就会采取一个行动来响应，这都取决于对象的性质和行为。我们前面已经提到过在跟踪和场景更新中存在的滞后和延迟问题。跟踪的精度依赖于以下几个方面：几何表示的精度（例如对象在空间中位置和方向的误差程度）、采样频率（每一秒样本的数量）、以及数据传送和解释的速度。

存在的最后一部分内容就是参与者的本体感受。所谓本体感受指的是内在的（无意识的）心理模型，即关于人对自身当前状态和倾向的模型。似乎我们有两个身体，一个是真实的肉
23 体，另一个是对应的精神躯体。为了要知道你的左脚现在在哪里，你并不需要去看——你的本体意识（连同其他触觉信息）将会告知你这些。有时本体感受会与真实人体倾向不一致（举例来说，一个人可能弯腰驼背然而感觉自己正站得笔直——Sacks（1998）给出了其他一些各种各样类似行为的例子），但是正常情况下它还是身体状态的正确表现。它和平衡感知一起提供了有关躯体、四肢以及它们当前运动的内部心理模型。

存在的一个必要条件是传感数据和本体感受彼此匹配。一个参与者感觉到在移动自己的手臂，相应地也看见化身的虚拟手臂在移动。如果手臂在向一处火焰移动，理想情况下此时会有热的感觉。沉浸感越高且传感数据和本体感受之间的匹配程度越大，存在的程度也就越大。

在虚拟环境中行走是一个非常好的例子。在虚拟环境中行进可以以真实的走动方式（就像在UNC大学实验室中用天花板上的跟踪器跟踪的例子那样）来产生，也可以用一些代替方式来实现，例如按住鼠标上的一个按钮等。在第一种方式中，参与者有步行的所有本体感受和相应的视觉变化的传感数据。在使用鼠标的情况下，视觉变化表明人在运动中，但是本体感受是参与者仍然站在那按着鼠标按钮。有的时候匹配视觉和本体感受信息的要求是非常强烈的，以致于参与者一旦按住鼠标按钮、感受到视觉的变化，就禁不住会开始真的走起来。有关行走问题的讨论可以参见Usoh et al.，1999。

现在让我们回到在这一小节开始处曾提到的1999年在 UCL所做的实验。随机地选择了20个人，任意将其分为两个组，一组10人。每个组都被告知要去寻找藏在实验室某处的一个红色盒子。其中一个组是在真实的实验室中完成该任务，另一组是在模拟实验室的一个虚拟环境中完成这个任务。在搜索任务结束时，每一个测试者都被要求填写两份完全不同的问卷，这些问卷的目的是要得出有关他们在搜寻期间的存在感知情况（见Usoh et al.，2000）。其中的一个调查问卷是要针对下面这些主题的六个问题得出结论的：即在实验室中存在的感觉、在多大程度上它能变成一种主导的真实、在多大程度上实验室空间被当成了一个真正的地方而不是一幅图像。第二个调查问卷（参见Witmer and Singer，1998）是由32个问题组成的，主题是有关可能与存在相关的一些因素，如在多大程度上人感觉到能自我控制、与视觉环境关联的感受等等。实验的结果是十分引人注意的，两个小组关于存在的程度的回答几乎都是一样的——这也就是说，无论是只经历真实实验室的人，还是那些只经历过虚拟实验室的人，在平均意义上所报告的存在程度是完全一样的。

彩图1-21给出了真实实验室和虚拟实验室的图像。真实的实验室远比虚拟的要“真实”得多——充满了细节和杂乱，没有办法使一半被试者将虚拟实验室误以为真。来自实验者的结论表明调查问卷本身不能区别真实的和虚拟的体验——换句话说，测试者都认为这些体验是独立的，虚拟世界中存在的程度并不是与在真实世界中的存在程度做比较得出的，而是将 24
那种特别的体验与某种理想情形做一些比较。另外的一个结论是测试中的主体通常都试着去找出测试者问他们这些问题的含义。如果当感知数据明显是只来自真实世界的时候，他们被问的问题是有关在真实世界中的存在感觉，因为他们很明显是实际地在那里，所以他们可能按照自己的理解重新解释问题——例如他们感兴趣的程度、舒适度或是投入程度。然而，有一点仍然是十分有趣的，那就是尽管所设计的模拟实验室的虚拟环境非常地简单，但对随机选择的这样一群人来说，在该虚拟环境中的存在感知程度总体来说同真实实验室是一样高的。

在整个人类历史过程中，从来都是我们所见的就是我们所在的。更精确的形容比如当我们转动头部和身体或者是当我们到处走动的时候，毫无疑问我们所接受的传感信息都是在描述我们的位置、我们在哪里。沉浸式虚拟现实打破了这个模式——我们所看见的不是我们所在的地方，而是由计算机所产生并显示出来的。我们所在的地方已经被那些显示器隐藏起来了。虽然在这样的一种经验中我们确实知道我们的真正所在（例如当我们在实验室中戴着一个与计算机相连的头盔显示器的时候），我们自身的传感系统以一种完全相同的方式处理从虚拟世界来的视觉和其他的感知数据，就像处理“一般”感知数据一样。我们的感知系统基本上不知道我们正看着的环境是一种幻觉环境——因此我们无法控制住自己做出真实的反应，就像我们在相同的真实情景中所做出的一样。一个虚拟的深坑在我们自治的神经系统中刺激起相同的反应，如同面对一个真实的深坑——无论我们对自己重复过多少遍“我知道那不是真的”。

我们所看见的就是我们所在的地方，但是我们在哪儿是存在于我们的大脑中！小说的读者自动且不加思索地将文字翻译成他们大脑中一个内在的情节，并在那个情节中对事件做出响 25
应。电影观众和游戏玩家也把外部的视觉和听觉数据转变成一个内在的心理情节——只是线索更加丰富，更加贴近在现实中曾经体验过的传感数据。在沉浸式虚拟环境中，同样的事情也在发生——在那里传感数据几乎同真实世界中的一样让人无法抗拒。线索更加直接地与我们头脑中的内在情节相关联，相比于文字或电影，只有在这种情况下，从媒体的形式到我们的内在模型之间的转换数量要少得多，同样我们也能通过包括整个身体的行动来改变正在进行的事件序列。另外，在这种情况中，我们看见了很多在运动中所看到的视觉现象，就像在现实世界中一样。当我们移动头部时，不在视线范围内的对象进入视野，而看得见的对象会从视野中不断消失。对象之间的可见性关系在变化——一个对象完全覆盖另外一个对象，现在只覆盖了一部分。在那里能做完全的透视缩短，随着我们视图的改变而适当调整。最重要的是这里存在着视差效果——当我们从一边向另一边移动头部时，看起来比较靠近我们的对象移动速度比离我们较远的对象要快，且近处的对象会遮挡远处对象的背景。

在现实世界中，从外部世界来的传感线索与我们内在的对现实的心理表现形式相连接，我们从这种内在表现出发来动作，而不是完全由传感数据驱使和确定。人们能在虚拟的实验室中存在是因为他们能看见桌子、椅子、灯、墙壁，而且能像在日常生活中一样穿越其间，尽管这样的虚拟环境表现是相当简单的。他们与头脑中的环境模型进行交互，由虚拟环境的视觉采样和那些抽象描写的虚拟对象来激活这个环境模型。如同我们看到一些直线段进而认出一个3D立方体，或者是看到“头部”、“两只手”、“两条腿”，于是认出那是一个“人”一

样，在虚拟环境中我们感知那些可识别的对象，这些构成了我们所沉浸的空间，让我们有了存在感。但是我们也确实有了像在现实中一样的行为。因此从这个角度来看，在虚拟世界中能产生与在真实世界中一样的存在感就不是什么令人惊讶的事了。

上述的讨论看起来好像是推测性的，但我们有非常强的科学证据支持这样的观点：即虚拟现实能发挥作用（引起存在感），这是因为它能像现实一样完全正确地触发相同的感知机制。我们将在下一小节中讨论这一点，在那里我们会接触到L.Stark 教授的思想（Stark，1995；Stark and Choi，1996；Stark，Privitera，Yang，Azzariti，Ho，Blackmon and Chernyak，2001），以及Richard Gregory教授的思想（Gregory，1998b）。

1.6 虚拟环境如何工作

"虚拟现实之所以能起作用是因为现实是虚拟的。"

Lawrence Stark教授，加州大学伯克利分校

现实是虚拟的

当我们四处观看时，我们体验的是一个时空连续系统，构成了一种非常高分辨率的视觉
26 流，通过这种视觉流我们看到了一个 3D 的世界。无论往何处看，我们都能看得非常清楚，每个点在我们的视觉中都是完全聚焦的，在视觉流中没有空间间隙，而且在时序上也是连续的，时间上没有间隙。然而这是一个幻觉（Stark，1995）。让我们思考一下这是为什么。

固定你的视线在当前环境的某个点上。注意到你所注视的那一点是固定的，其周围的一个小区域有最高的清晰度，不只是有高分辨度，而且色彩清晰（对正常视力来讲）。当你保持视线在这一点上时，你会意识到更远的区域在逐渐进入你的视觉范围。注意在这些区域中你看到的东西都不清晰，而且能感觉到颜色都很模糊。让我们研究一下图4-4，并阅读相关的段落，尤其是第一段的内容。注意，视网膜是图像被投影到的地方，包含数以百万计的对光敏感的单元（柱状和圆锥状），这些光敏感单元将光能转换为电信号，并通过视神经传输到大脑的视觉皮质。视网膜上只有一个极小的区域能产生0.5度到2度的高清晰度视觉（包含小凹的斑点）。视网膜图像只有在这个小的区域中才是清晰的。斑点是视网膜上的一个很小且浅的凹陷（大约6毫米乘7毫米），它所包含的光感应单元都是具有高分辨率和色彩感应类型的（圆锥体）。凹陷（大约1.5 毫米乘1.5毫米）是一个紧位于斑点后面的区域，它上面的圆锥体感光器的密度最高。这可从图1-22中看出。光通过眼睛的光学部件被聚焦在视网膜上的凹陷处，产生正前方的一个高分辨和色彩清晰的视觉图像。

所以我们只对视野中的一个小区域有敏锐的视觉能力，但是好像是在所有的地方都能看得非常清楚。这是因为我们的眼睛是不断移动的，当然，无论我们往何处看，那里的情景就会进入凹陷，因此被看得很清楚。视觉是一个采样过程，由交替眼动（通常叫做快速扫描）序列和定影所组成。定影将信息带进凹陷之内（当然也带进整个视网膜）。当采样环境时，眼
27 睛从一点到另一点移动完成快速扫描。每秒钟大约有三个这样的定影。

举例来说，当你走进一个房间的时候，来自这个过程的采样信息实际上是比较少的。通过对整个房间的扫视，你已经对整个房间有了认识。首先我们会建立有关该房间是什么类型的房间这样一个概念：卧室、厨房、起居室、教室、实验室等，更多情况下你已经知道了这些。每个房间通常都有四面墙壁、地板、天花板、一扇门、和一些窗口。你看见了这些，也看见了各种房间内相应的一些典型事物，于是你已经"看到了整个房间"。

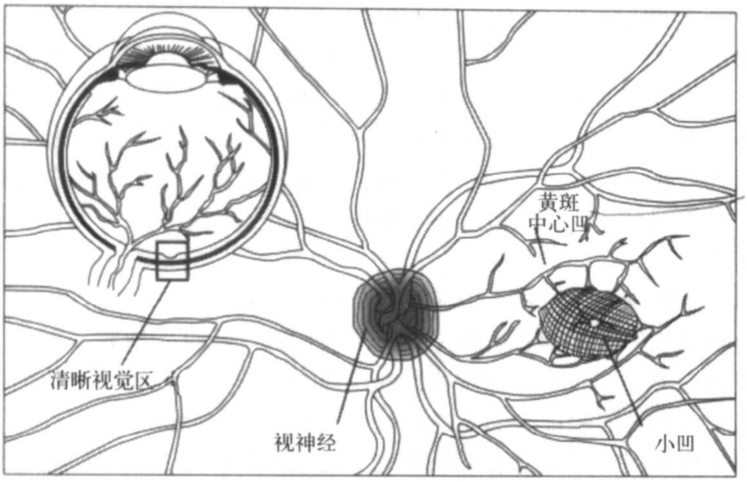


图1-22 高视觉区域的小凹

要确认你并没有“看清整个房间”是容易的，请试下面的一个简单实验，也许你下次进入一处你并不非常熟悉的地方。就像一般情况下我们通常是环顾左右以建立整体印象，假设它是一个房间，可能是图书馆中的一个房间，也可能是一个办公室或其他什么地方。现在我们建立一个心理滤波器，对你自己说“我想要看见红色的”。当你再一次环顾房间时，如果没有努力去找寻红色的东西，而是仅仅有意图去找到红色的东西。很有可能你的眼光好像着魔似的被吸引到红色的对象上，预先你一点也没有注意到它在那里，或根本没有注意它是红色的。现在让我们再一次重复相同的实验，但是这次使用另一个滤波器：“我想要看见绿色的”——同样，你或许对环境中绿色对象变得特别敏感，也许这些对象上次你根本没有看到。你甚至可以用更抽象的意图对自己重复相同的指令，如“我想要看见圆形物品”、“我想要看见矩形物品”、“我想要看见水平的斑纹”等等。最后面的一项会特别有趣：例如，你可能突然发现在你以前从未注意到的物体表面或边之间的对齐关系，即使这个地方你相对来讲比较熟悉。突然间你发现物品的边线像是排成一列，或者是你注意到你以前从未注意的一些器具上的水平网格或图案。有许多方式去观察一个环境。你能通过设定特殊指令改变知觉滤波器的方式来改变你观察的方式。你甚至能通过设定非常不寻常的滤波器改变该实验。举例来说，设想你是在屋内，或者是在一列地铁上，你可能说：“我想看见树。”——突然环境中的某些景物将会以出人意料的方式变成了树的形状。

*扫描路径*是交替快速扫描和定影的序列，当我们观察环境或照片的时候，这种序列一遍一遍地重复着（Stark and Choi，1996）。大约有 90% 的时间是在定影中花费掉的，再一次访问场景中感兴趣的相同地方，好像是要重复地确认某个概念。举例来说，“是的，这是个房间”，然后反复检查感兴趣的房间中的不同地方。然而，你看到了整个房间只是你的幻觉——许多细节甚至一些总体特征可能都被你忽略掉了。实际上你所注意到的只是不同之处。如果是一间不寻常的房间，你很有可能将会变得迷惑起来，对那些使你迷惑的东西给出多种不同的解释。 28

因为我们知道一个房间有四面墙壁、有地板和天花板，总是呈现出一个立方体形状，要想改变这种根深蒂固的假设是极端困难的。

彩图1-23显示了两个年轻的孩子站在Ames Room里，该名字是以一著名眼科医师Adelbert Ames，Jnr 命名的，他在1946年首先装配了该结构。它给人们的第一印象是它看起来

很像一个普通房间。第二个印象是孩子尺寸间的巨大差异。怎么会是这样的呢？在以往的经验中，我们从未遇到这样一个奇怪的房间，它的地板是倾斜的，是个非立方体形状的，墙壁之间不构成直角。房间的这种构造方式（Gregory，1998a）使得从这个视点看给人一种“正常”外观印象，但是事实上它一点也不是这样。地板是倾斜的，墙壁不以直角相交。双胞胎中大的那个离我们的视点要近得多。然而，因为我们的视觉系统知道它是一个房间，我们宁愿坚持认为看见了两个不同尺寸的孩子这样不可能的事，而不愿相信一种更可靠的假设，即房子的空间结构是扭曲的⊖。

因此首要的一点是我们的视觉系统通过扫描路径序列来采样环境，一遍又一遍重复地返回到相同区域，确认我们内部的心理模型，有关我们设想看见的内容。事实上，我们所看见的可以说已经在脑海中了，我们正在用我们的想像力去看，从环境中活动地扫描所见的只不过被当作了证据和对假设的验证，而不是直接导致我们看的原因。我们看到的是我们想像中的东西，并使用采样得到的视觉信息来确认它。通过不断检查和核查环境这样一种方法来确认它，找寻证实它抑或导出矛盾的证据。扫描路径本身是受到我们内在心理模型的驱使的，而不是受外部世界驱使的。内在心理模型告诉我们自己在哪。因为我们的模型本身确定了我们的兴趣区域，它构成了要验证的部分内容。改变模型则扫描路径本身也会跟着改变。

请看图1-24——你第一眼看见了什么？是一只鸭子还是一只兔子？无论你看见了什么，试着去给出另一个解释。注意在对两种解释的响应中你的无意识眼动。你的眼动是被你内心的模型所统治的，你内心的模型告诉你什么是你正在看的东西——眼动的改变完全依赖于你的决定。可以以完全不同的一种方式来看这幅图像，这依赖于你是把它当成一只鸭子还是一只兔子。“外部世界”——即该页上的征像——当然是无变化的，因此它们本身不能够导致两种不同的眼动模式。

29 图1-25给出了有关不明确图形的另外一个非常有名的例子。第一眼看去，你可能看见一个尖下巴的老妪，或者是侧向的年轻女子的头部。

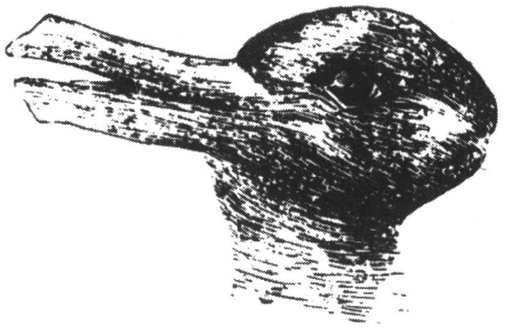


图1-24　是只鸭子还是只兔子？



图1-25　是年轻女子还是老妪？

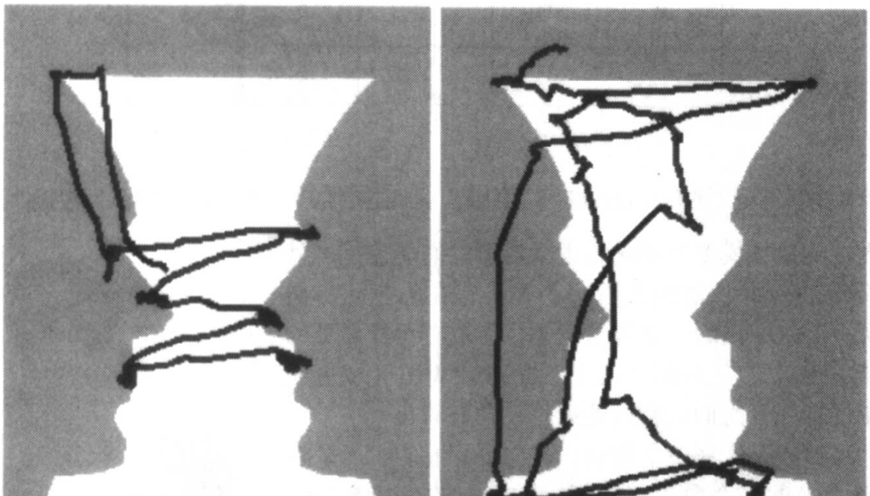
⊖　要想看到更多这类异常，请访问加利福尼亚Santa Cruz的Mystery Spot站点，http://www.mysteryspot.com。

你能改变对图的解释吗？如果是的话，再一次注意你的眼动模式是如何改变的。

图1-26给出了另外一个著名例子：你可能看到的是一个花瓶，或者是两个彼此面对的脸部侧影。Stark 等（2001）使用这个图的一个变形并记录了主体的扫描路径，这个主体可能被给予提示说明图中是一个花瓶，或者说明它是两张侧脸。图1-27a给出了在主体被告知图像是两张脸时的扫描路径。图1-27b给出了当主体被告知图像中是一个花瓶时的扫描路径。注意扫描路径在这两种情况下是完全不同的（看两张人脸的方式与看一个花瓶的方式是不同的），而且在统计意义下测试者的两种扫描路径间的差别是相当大的。显然扫描路径是受我们的自顶向下的认知模型所驱策的，而不是仅仅通过我们正在观察的事物，认知模型描述了我们将要看到的东西。因为，很明显对于这些不明确的图形来说，根据我们所观察的事物来扫描会是很确定的。 30



图1-26 是花瓶还是两张侧脸？

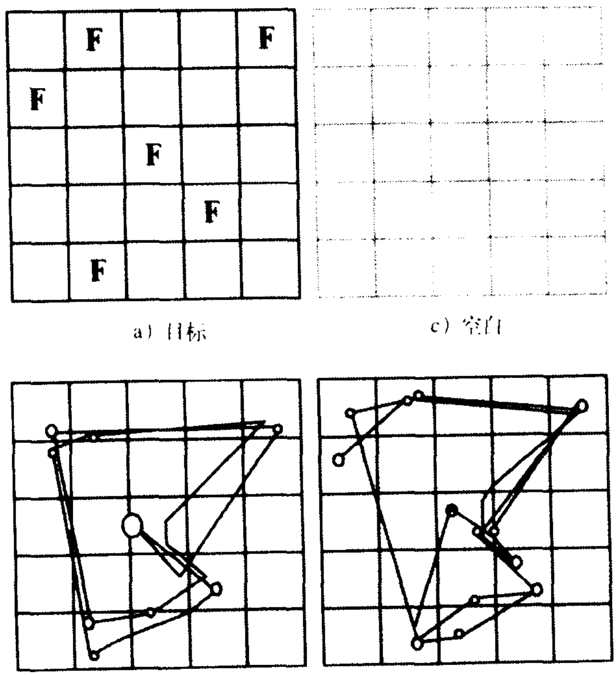


a）两张脸的扫描路径　　b）花瓶的扫描路径

图1-27 关于两张脸和花瓶解释的扫描路径

图1-28给出了来自另外一个实验的例子，是由Stark和同事共同完成的（Stark et al.，2001）。主体注视在一个网格中的字母集 20 秒钟，扫描路径被实验者记录下来。例子中的目标在图1-28a中给出，对应的扫描路径由图1-28b给出。他们重复这一实验两次，第二次注视目标7秒钟。然后他们被要求去注视一个空白的网格（图1-28c）并想像先前的目标。同样，扫描路径被记录下来（图1-28d）。图形说明了两种扫描路径之间的相似性。通过对整个实验数据的分析，我们可以看出，当注视目标时扫描路径之间并没有很大差别，而且对应的扫描路径没受任何实际物理刺激的作用——换句话说，这些扫描路径的产生是通过想像力完成的。因此，证据告诉我们，我们从外部世界中所看见的事物，如扫描路径所指示的，不只是简单地受外 31

部世界中的“物质”所左右，而是受到我们内在认知模型的控制。



a）目标　c）空白

b）目标的扫描路径　d）想像力目标的扫描路径

图1-28　想像力的目标扫描路径（由加州大学伯克利分校的Lawrence Stark教授提供）

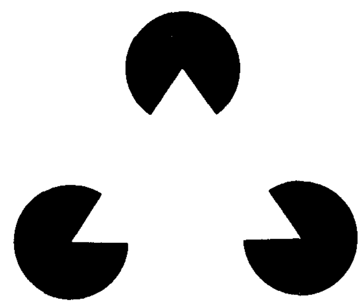


图1-29　Kanizsa三角形

我们内在的认知模型是如此有力以致于它们能盖过“外界”的实际存在，并执着地去看一些并不存在的东西。图1-29给出了有关这一点的一个著名例子。你最有可能看见的是在三个碟片之上的三角形，由于三角形较周围区域更明
[32] 亮。你甚至能在碟片之间的空白处看见三角形的边。

由于已经形成了这样一个假设，即你正在看的东西是在三个黑色碟片上的一个白色三角形，所以你的感知系统坚持认为有一个三角形在那里。这是关于所看到的东西的一种比较合理的解释，而并非是三个带有三角形缺口的碟片。我们的感知系统补充了三角形的三条边，甚至在空的地方中，我们似乎无法看不见它，即使内心深处知道那里并没有，我们这种内在模型就是这样强有力。

这是认知模型支配作用的相对“高层”的例子。这里还有另外一个比较低层的例子，它清楚地说明了感知系统如何精确地填充那些似乎丢失了的信息。让我们再看一次关于人类眼睛描述的图1-22和图4-4。你会注意到信号是由视神经从视网膜被送到大脑的视觉皮质的。在视神经与视网膜相交处没有光敏感的接收单元（没有柱状或圆锥状光敏组件）。所以两个眼睛在这些区域一定会出现视域间隙。然而，你并没有注意到这些间隙。通过一个简单的实验我
[33] 们就可以注意到这点。伸直你的左手手臂，使你的手和食指指向右侧。闭上你的左眼，让你的食指指尖和你的鼻梁对齐。现在同样地把你的右手手臂伸直，让你的右手食指指向左边。

你的右手臂最初要与左手臂保持大约45度角。现在闭上你的左眼，盯着看你的左手食指，慢慢地将右手食指移向左手（图1-30）。在某一时刻，你的右手食指指尖将会消失！当你继续向左移动你的右手时，你右手食指尖将会重新进入视域内。视域间隙是与你右眼睛视网膜上与视神经相连的那块区域相对的。



图1-30 怎样看到视神经所留下的间隙

那么我们为什么通常没有注意到由视神经引起的视觉间断性呢？这是因为视觉系统（或许在视网膜本身处理层次上）的一些基本图像处理工作是基于对周围区域信息的整合来推理得出视觉体验应该是什么样的。这在图1-31中给出了说明，该图由 R.L. Gregory 给出。固定你的视线在图中右下方的黑色斑点上。你将会注意到在水平线中的圆形间隙很快消失，你甚至 34
能看到水平线慢慢地传播自己，穿过间隙直到把间隙清除掉。我们的视觉系统形成了这样的一个假设，即水平线在各处都更像是连续的，而不是存在一个料想不到的间隙——于是视觉系统除去了这个间隙。

有另外一个非常重要的例子，关于视觉系统是如何用高层认知信息替代传感数据的。这叫做*尺寸不变性伸缩*（Gregory，1998b）。同样，这可以被 R.L. Gregory 给出的另一个简单实验说明。伸出两只手，一只手臂尽量伸直，另一只手伸到一半的位置，让每只手都朝向你。让你的两只手之间保持一个较大的距离。闭上一只眼睛，比较你两只手的大小——它们看起来是一样大的——它们都是实际手的大小！现在让我们将它们彼此靠得近一些，此时相比之下，一个看起来像是一个儿童的手了（图1-32）。我们的视觉系统对已知距离较远的对象在视网膜上的投影区域补偿较小，或者这种补偿直接根据图像的透视线索得出。看靠近你的人们和在 35
遥远处的人们。他们看起来大小是相同的——都是正常人的尺寸！然而在视网膜上占据的区域却非常不同。如果你在一条普通的街道上观察一个房子或者一幢建筑物，它看起来很大（即使它所占的视网膜图像的面积相对较小）。然而，如果你在一片房子的上空中俯瞰，它们看起来很不真实，像一个玩具村庄。我们习惯于从地面上看这些建筑物——我们的感知系统对从上面看这些对象没有什么经验。因此，尺寸不变性伸缩使物体看起来较之真实传感信息形成

在视网膜上图像的尺寸发生了变化。然而，一张通过照相机拍摄的照片不包括不变性伸缩，所以当我们检查图1-32时立刻能看出图像实际上具有不同的尺寸。

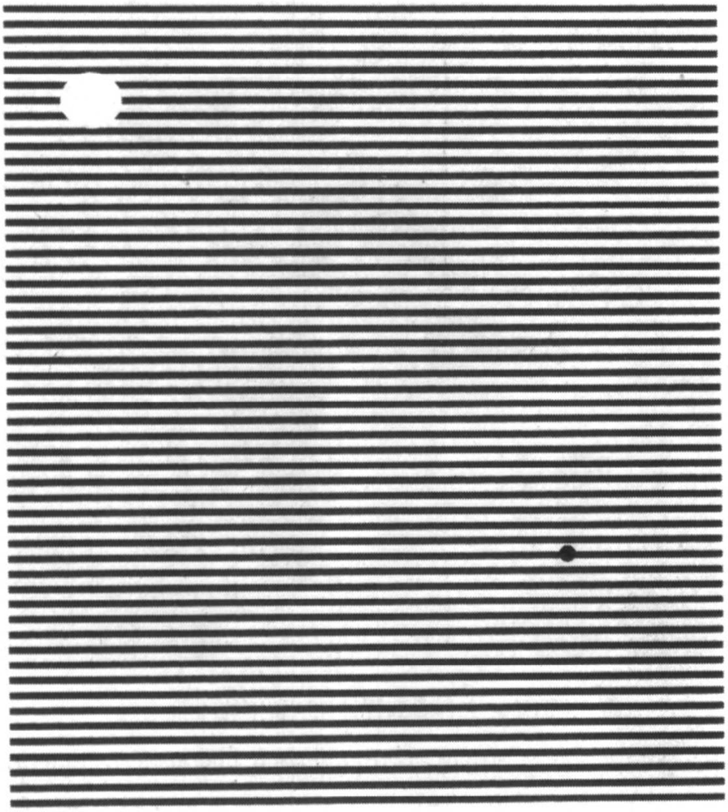


图1-31　当我们将视线固定在斑点上时碟型空隙不见了

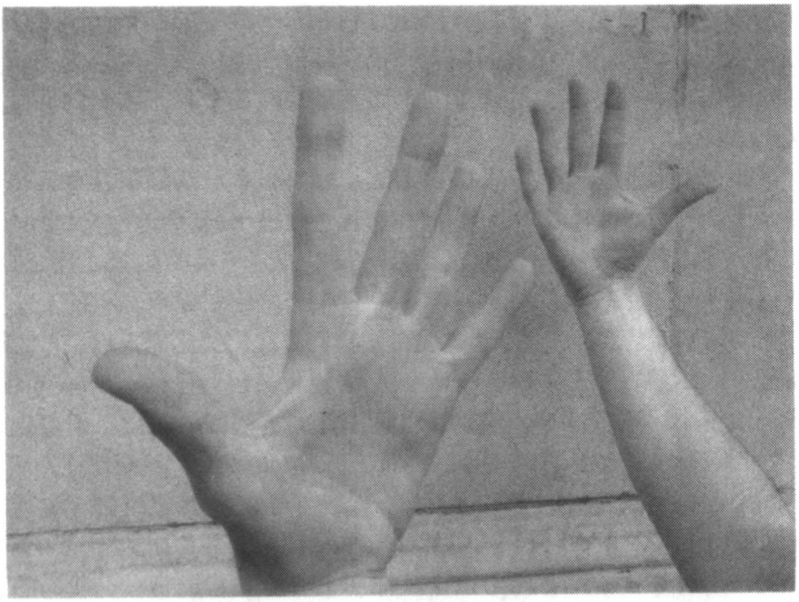


图1-32　手的大小

在另外的一个由R.L. Gregory给出的实验中，我们能体验不变性伸缩的直接效应(Gregory，1998b)。对一盏明亮的灯注视几秒钟，为的是要形成一个好的残留影像。然后将目光移向房间里的一面墙上，注意残留影像的尺寸如同是在墙壁上投影。现在当你朝墙移动时，残留影像将变得越来越小，而当你远离墙时，残留影像将变得越来越大。这是一个直接证据，证明图像的外观尺寸是随着距离而改变的，即使视网膜上的图像大小保持不变。深度是不变性伸缩所基于的一个线索，另一个线索是透视。我们将在下一小节中讨论。

在三维中看

通过双眼我们看见一个三维中的世界。每个眼睛接收到世界的一个略微不同的视图，视觉系统融合这两幅图像形成三维空间的立体视图。这两个图像间存在的差别称为双眼视差。当然明白这一点的最简单方法就是你现在朝所处的环境中的任一个地方看去，而且交替地用两只眼睛看，一只睁开时则让另一只闭上。注意图像是如何水平地从左到右然后从右到左地来回切换，这就是视差。需要注意的还有对于比较近的对象视差很大，对于远处的对象视差则较小。

体验它的另外一个方法是使用彩图1-33a。拿一页A4或美国信纸大小幅面的硬纸，让纸的一个短边与两幅图像的中线重合，保持纸面与书页垂直。现在将你的鼻尖放在纸的另一个短边上。检验你的左边眼睛只能看见左边的图像（闭上右眼），而且同样地你的右眼睛只能看见右边的图像。现在睁开双眼，允许两幅图像融合在一起。看起来好像有两张纸，在它们之间有一个完整的立体3D图像：一个男子在向你跑来，隧道在不断向后退去。这应该是个非常强有力的效果。

同是这张图，它还可以用来说明视觉的其他两个非常重要的方面：调节和收敛。首先，再看一次图4-4，注意这个透镜。显然在眼睛中最强大的光学能力来自角膜，它装满称为晶状 36
体的液体。透镜用来精细调节以便让相关点完全聚焦在凹陷上。透镜是受睫状肌控制的，睫状肌通过动作来控制它的厚度，从而控制它的光学特性。调节是调整透镜以便让场景中的点对准焦点的过程。

在另一方面，收敛是眼睛向内旋转以便让附近的对象进入焦点；或者向外旋转，使视线趋于平行，以便让远处的对象进入焦点（图1-34）。收敛在体验场景深度时是3D视觉一个非常有力的线索。

在正常的视觉中调节和收敛一起工作。然而这不是生理上必须的，而是在生活中习得的。很容易说明这样一件事，即在一产生完全幻觉 3D 视图的虚拟现实中，调节和收敛之间的关系已经不复存在。再看一次图1-33，这回你持续地盯着正在跑动中的男子，然后沿着隧道将视线连续地移向深处。当你固定视线于场景中虚拟距离上的每一个不同部分时，你已经体验到了收敛的改变。但是调节呢？这当然是没有改变的，因为为了要清晰地看见图像你一定是全神贯注地盯着有图像的这张纸。因此收敛是改变了，但是调节确实没发生变化。这一点得到了那些使用头盔显示器等虚拟现实设备的人的证实，他们使用这些设备感到眼睛不同程度的疲劳，但是没有发现对视觉的长期影响。

请闭上你的一只眼睛，并环视左右（这种实验已经做过很多次了!）。多么令人惊奇，世界仍然是3D中的世界！为什么会是这样的呢，是因为我们在前面解释清楚了3D体验是源于双眼视差 37
的结果吗？电视和电影给我们如此强烈的3D印象，我们甚至根本不去提醒自己实际上我们所看

到的不过是个纯粹的 2D 图像？为什么我们会害怕三维怪物？为什么三维的惊险读物会使我们恐惧，而三维喜剧会使我们发笑？这是因为除了双眼视差线索之外三维场景还有许多其他线索。正如我们前面已经讲到的，当我们闭上一只眼睛看真实世界的时候，收敛仍在工作，它自身就能提供强有力的深度暗示。其次，在这个环境中仍然有头部运动视差，这一点我们早先提到过——当我们移动头部时，比较近的点移动比较快，较远的点移动比较慢，这是场景深度的另外一个线索。在电影和电视中既没有收敛也没有头部运动视差，但是却有移动视差。离镜头近的对象移动速度比离镜头远的对象移动得要快，这仍然是一个强有力的视差深度线索。3D 场景的二维图像有额外引人注目的深度线索，而我们也是在不断地来回考虑它们（Hodges and McAllister，1993）。

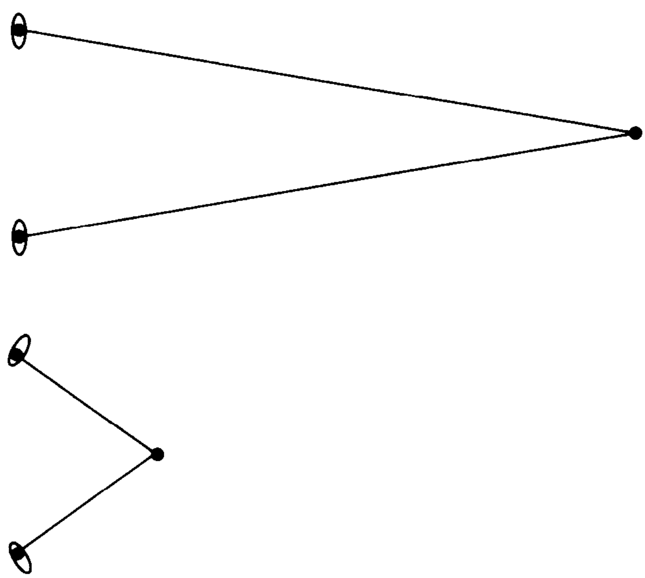


图1-34 收敛：对于近处的物体两眼向内旋转

线性透视。我们注意到在视网膜上形成的图像尺寸是反比于对象离观察者的距离的。这可由图1-35说明。我们对眼睛作个简化；特别是，我们假设光不是经过瞳孔、角膜和透镜，而是通过一点(O)进入视网膜的，然后在（平面）视网膜上形成图像。在计算机图形学中 O 这一点通常被称作投影中心，而视网膜被称为图像平面或视平面。我们给出AB和CD投影的“侧视图”。如图所示，AB投影到ba，CD投影到dc。AB 和CD具有相同的尺寸，但是它们的投影尺寸显然与它们到O点的距离成反比。同时注意所得的投影是颠倒的图像。

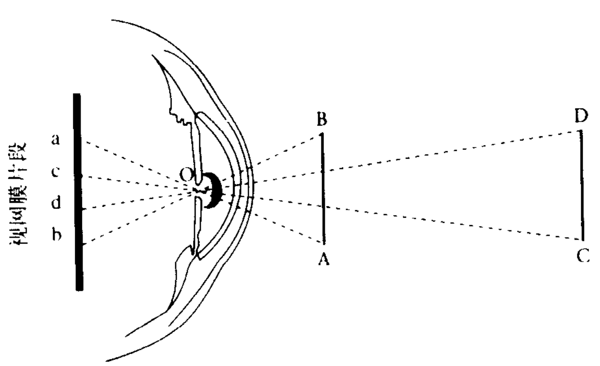


图1-35 线性透视

透视提供给我们一个非常重要的深度线索。这可由图1-36说明。两个发灰的垂直线具有相同的高度，但是一个看起来要比另一个大很多。此时透视给人一个深度的幻觉，所以一个 38
看起来好像离我们很远，回想一下不变性伸缩，视觉系统通过使它的外观尺寸变大来补偿不变性伸缩（参见Gregory，1998b）。

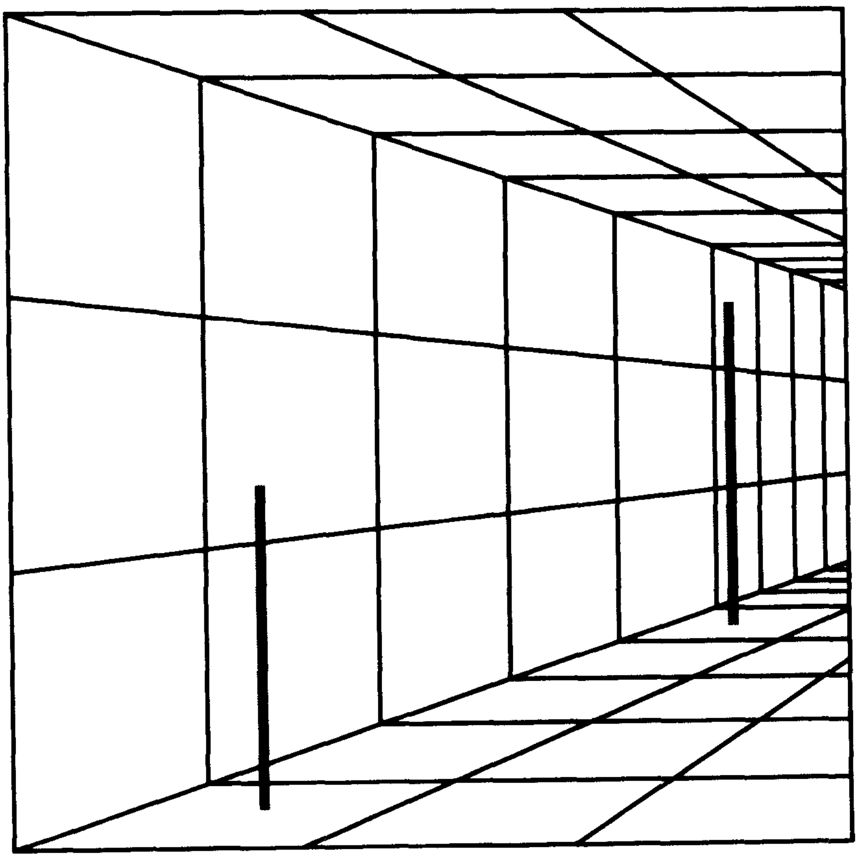


图1-36 线性透视的错觉——粗线具有相同的高度

相同的效果可以从图1-37中看到。在这个极为简单的图中我们产生了沿着铁路线向前看的感觉，或是道路在向远处延伸的感觉。因此位于上方的水平线显得比下面的水平线要大很多，这是因为透视给我们的印象是上方的那个水平线处于远方，因此它一定是比实际看到的要大许多。

图1-38给出的是著名的Muller-Lyer错觉。虽然两个垂直直线具有相同的尺寸，但是它们看起来好像左侧的那个要比右侧的那个小很多。R.L.Gregory（Gregory，1998b）解释说，这
是透视诱导的尺寸不变性伸缩。左侧的形状给人的印象是个朝向我们的角落（例如，从建筑 39
物外看两面墙壁拐角）。右侧的形状给我们的印象也是一个墙的拐角，但它却是远离我们的（从房间内部看墙角）。两个垂直线条的视网膜图像的尺寸是相同的，但是透视在诱导表观深度、尺寸不变性伸缩导致了垂直直线长度产生了显著的差异。那个朝向我们的拐角形状因此看起来要比远离我们的那个右边的形状“近”得多，所以右边的那个形状中的垂直直线感觉上被放大了。

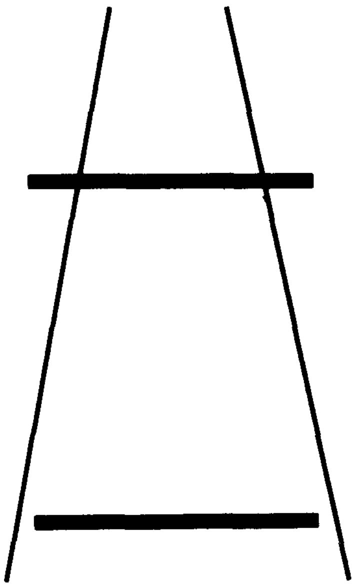


图1-37 另一种Ponzo错觉

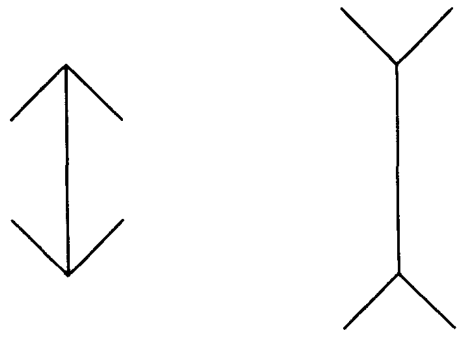


图1-38 Muller-Lyer 错觉

纹理梯度。让我们审视一下图1-13，尤其看一下水面。水的纹理愈靠近视点越显得粗糙，越远处越精细。图1-39给出了一个更抽象的例子。同样地在这个图中没有透视直线，只是布
40 置了一些小圆点，越靠上面越密且小。然而，整个图给予我们的印象却是一个延伸向远方的景象。感知心理学家 J.J. Gibson 对这样的纹理梯度做了一些研究，说明为什么它们在对真实世界场景尺寸和距离判断上是至关重要的（几乎总是充满着多重纹理）(Gibson，1986)。

阴影和明暗处理。阴影在现实生活中能大大地提高对深度的感知（Puerta，1989；Gregory，1998b)。阴影不仅能传达有关对象形状的信息，而且能传递在场景里的深度关系信息。对象的阴影可以看成是对象的另外一个视图——由光源产生的视图。因此这是一种额外的有关对象在空间中位置的关键信息，因为在那里不但有关于观察者的参照物，而且观察者能看见有关另外的一个参照物即光源的信息。阴影也提供每个对象与周围表面间空间关系的直接信息。图1-40给出了一个简单的例子。左边的立方体显然是位于地面上。右边的立方体不是在地面上，可能比左面的那个离视点要更远一些。

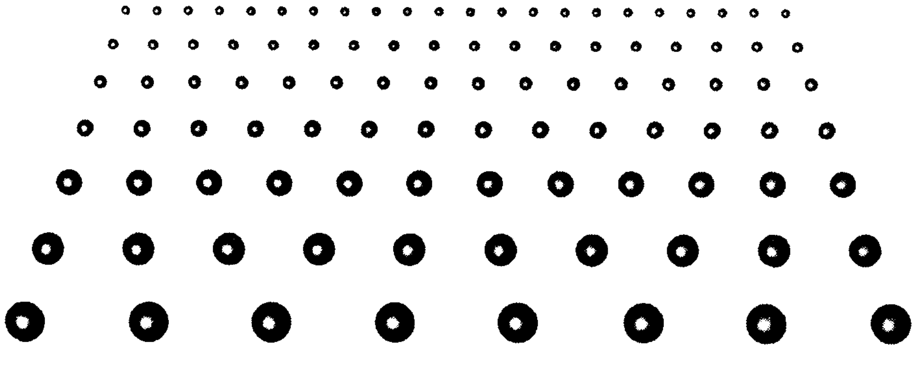
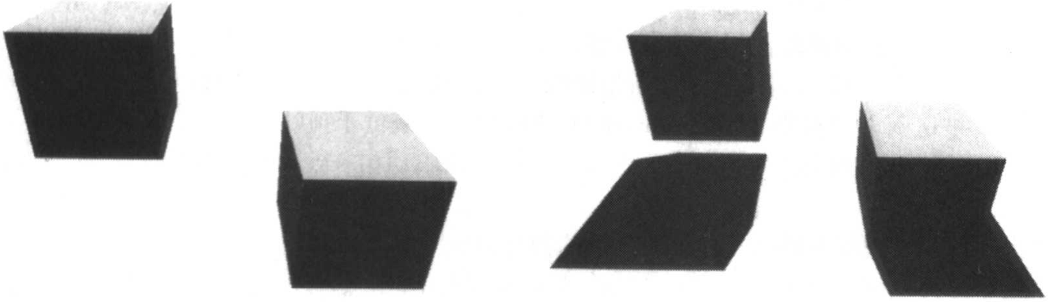


图1-39 纹理梯度的抽象实例



a）不带阴影的两个方盒

b）带有阴影的两个同样的方盒阴影有助于理解两个方盒之间的关系

图1-40　阴影深度线索

遮挡。我们在有关视差的上下文中曾经提到过遮挡效果——当你转动头部，或当对象移动时，它们之间的关系在改变。对象进入和退出视图，它们与视域的关系以及它们彼此之间的关系都在发生改变。当一个对象部分地遮挡其他对象的时候，它给人们一个清楚的指示，那就是该对象比较靠近观察者。这种效果是非常强的，它是图1-29中Kanizsa三角形错觉形成的基础。 41

光照。如我们将在第3章中所看见的，光能量的衰减与距离的平方成比例。因此较远的对象与近处的对象相比，看起来颜色比较淡，同时比较模糊。而且，虽然计算机图形学对光的传播做了简化，假设光在真空中运动，在真实世界场景中当然不是这样的，而且大气影响颜色的外观，随着距离而逐渐变浅：从远处光源传出的光似乎变得更蓝。比较近的对象通常颜色是比较明亮的（其他方面都一样）。更远处的对象通常被大气效果所笼罩。我们在原始时期就已经了解了这种效果与距离的关系。在相应的地方它们提供了强有力的深度线索，尤其像在这幅画中，包含了本节所讲述过的各种深度线索。

时间和空间不变性

我们在时间上是连续地看物体吗？抑或在你的正常视觉中注意到存在中断吗？当然你几乎没发现过这样的中断。但这也是另外的一个错觉。注视其他人一阵子，视觉频繁地中断吗？回答应该是非常明确的："是的"——每一次某人眨眼都是一次中断，当然，在眨眼的瞬间人们什么也没有看到。既然你了解到眨眼的事实，这时可能就开始注意到你的视觉是频繁中断的。让我们放弃考虑这些，我们通常都不注意眨眼的事实，我们的视觉像是连续而不间断的。也有另外的一个视觉被中断的方法——每一次我们移动眼睛时，在我们的视野中都有一个大的不连续跳跃。而且，我们的眼睛几乎是无时不动的。是差别在刺激我们的视觉。如果我们能完全固定我们的视线，不久之后我们会什么也看不见，这是因为光接收器单元适应了。它很像环境中存在的连续背景噪声，例如时钟的滴答作响；过一会儿之后你就根本听不到它了。我们的感觉大致上适应了，不断地被差别所刺激。由于对眼睛快速扫视的抑制，我们不会注意到眼睛运动，这种抑制作用降低了快速扫视眼动期间的视觉灵敏度。

世界对你来说是不断地运动的吗？回答又是"不"——我们的视域看起来是空间稳定的。另外一个实验：闭上你的一只眼，用食指轻轻地触动睁着的那只眼的眼球。发生了什么事？——你的整个视域在向你手指戳的方向移动。但是为什么当你眼睛自然地移动时，它不动，而通 42

过手指外部推动时它才移动呢？

当我们连续移动眼睛使得凹陷在视觉场景中从一点跳到另一点的时候，该视觉场景在视网膜上就是从一个场所到另一个场所地跳跃。眼动肌肉所接收指令的放电“结果”在每次眼动之前被送到亥姆霍兹（Helmholtzian）比较仪，大概位于顶叶处。这个预计算提供了空间不变性的错觉。如果视网膜的图像运动与受控制的眼动相一致，那么错觉就存在（Stark，1995）。

换句话说，视觉系统执行一个预报性的预计算——如果眼动指令是在“x”方向上移动，那么为了保持空间域的连续性，用“－x”来补偿视在图像。这样的时间不变性的获得是依赖于对眨眼和快速扫描眼动的抑制效果，同时在空间上通过前瞻性预报估计由于眼动命令而引起的视网膜图像的位移量。

视觉记忆假设

我们所见的明显依赖于我们是在哪里看和如何看。扫描路径理论提供证据表明，我们所看见的东西主要依赖于我们的内部认知模型，或者说是用我们的想像力去看的。证据说明对一个想像中的对象的扫描路径与实际上看那个对象时产生的扫描路径是类似的。它也说明了扫描路径如何改变依赖于对我们所正在看着的事物的不同解释。例如图1-26中的那个不明确的图形。因此我们看见什么不是光能量进入我们眼睛的一个简单函数，而且还与我们所期待看到的事物有关。有关这一点的进一步证据就是，扫描路径既是与我们个性相关的，而且也是与我们所看的事物相关的，同时是不断重复的。从扫描路径因人而易的事实还可以得出另一个结论，即我们如何去看不只是我们所看着的对象的函数，它也依赖于我们个人的认知模型。而且，如果看主要是“外部世界”的函数，那么扫描路径将会覆盖整个场景。事实上情况并非如此，场景中总是只有很少的点被一遍遍地扫描过，每一次这样的扫描顺序基本一致。这是在看某一类对象时的主要特征，重复扫描的作用是一遍遍地证实它就是我们所正在看的对象类型，而不是其他什么类型的对象。

对任何一个特定的单一传感体验总是存在多重解释。我们的视觉记忆以最大概率迅速导向目标，这是基于多方面因素的，例如先验知识、暗示、经验以及周围的上下文。当然我们
43 很少看见孤立的对象，它们几乎总是存在于某个上下文中，这样的上下文对于我们应该如何解释它们提供了线索。在没有其他知识的情况下，一个位于厨房或客厅中的某一个对象可能既被当成一个微波炉又被当成一个计算机屏幕。在对象仍然不明确的情况下，视觉记忆试图重复地在多种假设之间来回转换。所以我们首先看见一个，然后再看一个，这仿佛是一个循环，每一种解释都有其自己相应的扫描路径类型。

现在让我们做个实验。注视图1-41中圆的中心，你将会看见三角形的一个排列。凝视圆的中心，该三角形的排列将换成另外的一个三角形排列。继续凝视该圆，你会感觉到这种排列的交换速度在逐渐提高。视觉记忆在两个可供选择的假设之间转变。

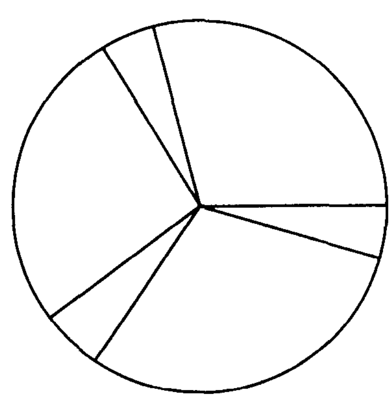


图1-41　二义图形：解释自然地切换

当某个人“在”虚拟环境中，他们将不可避免地会处理来自两种不同或者相互矛盾的信息来源的传感信息。典型情形是视觉信息只从虚拟世界中单独产生，比如使用了

头盔显示器的时候。让我们设想听觉信息也是从虚拟世界经过连在HMD上的耳机传过来的。然而，人毕竟是站在一个真实实验室的真实地板上，在自然的温度条件下，能感受到自然风的存在，有一条缆线连到HMD上，而且能感受到HMD本身的沉重。虚拟环境的参与者的总传感数据是来自虚拟环境本身和现实世界各种体验的混合。这类似于不明确图形的情形——相同的传感数据可能有多种解释，因而引起人来回地看以验证自己的认识。我们曾经讨论过（Slater and Steed，2000）在某个环境中存在需要在不同假设中选择，需要用不同的方式去观察和分析各种传感数据（有多种感知通道存在），不同的传感数据支持不同的存在假设，有的能证明你是在这个世界（虚拟的）中，有的能证明你在那个世界（现实的）中。在虚拟现实中的存在意味着传感信号的组织方式是使得参与者暂时经历一种 VE 世界里的真实，并对这个世界中的事件做出响应或引起某些事件。偶尔解释也会出现翻转现象，传感信号将会告诉你："我正戴着头盔站在实验室中。"

让我们再一次看图1-25。当你看出图中的老妇人的时候和你看出一个年轻女子的时候，你所注意的特征是截然不同的；或者说在这两种情形中你的扫描路径是完全不同的。同样， 44
当你在虚拟环境中存在时，你所注意的信号集与你在真实世界场景中体验时所注意的信号集是完全不同的。

举例来说，在后一种情况中你可能已经把注意力切换到触觉感知上了，而在虚拟环境中存在的那一刻你可能已经忘记了这种感觉。这种在 VE中存在的解释和在真实世界中存在解释之间的倒转对许多有虚拟环境体验的人来说都是有过的。要去研究的一个重要问题是去发现影响两种不同解释（在虚拟环境中存在和在真实世界场景中存在）发生概率的因素。

在这一小节中我们已经讨论了在 VE 中存在感知的一个必要条件是虚拟的感官输入刺激相同的视觉体验，这种视觉体验是自顶向下的，同时也是由认知支配的，就像来自真实世界的感官输入一样。Gregory （Gregory，1998b）注意到"感知与现实之间存在着一种微妙的关系，是我们意料之中的，蕴藏在我们的自顶向下知识结构和基于过去经验的横向规则里"。 45
Stark的工作指出我们是在用我们的想像力去看，来自真实世界的自下而上的感官信号是驱使我们的视觉记忆在多种可能假设之间选择的基本数据。扫描路径是内在认知模型的外在表现——人通过内在认知模型来分析视觉传感输入信号的特殊模式。

那么要引导视觉记忆基本上像在现实世界中处理信号那样在虚拟环境中工作，需要虚拟环境的（视觉）传感信号具备什么样的性质呢？当然输入信号的任何模式都能强制视觉系统产生这样的作用，尤其当这些输入信息是多通道的，例如阻断了各种相同感官类型的所有其他信号。人们能在随机图像中读出有意义的结构，例如在散乱的茶叶、云团，以及像在性格测试中被用到的著名的Rorschach墨水污点图像中。这里有一个例子（见图1-42），它给出了一个随机产生的多边形图案，你可以从中看出些有意义的东西。但是有哪些特殊的特征是传感数据需要具备的以便描写一个主观上有确定意义的图像呢？

在这一章中我们曾经讨论过很多可能性：通过线性透视、头部运动和运动视差、纹理梯度、由可见性关系和阴影等提供的深度信息等来描绘图形。对象表现需要一定水平的真实感，这种真实感只要能充分说明是要仿真的对象即可。漫画很重要，它突出了某些关键特征，这些关键特征能立即激发视觉记忆，形成它就是那种对象（而不是其他对象）的假设。当我们描写人脸时，非常简单地表现眼睛、鼻子和嘴就足够了。我们并不知道主要特征是什么，于是只好尝试去构造一个能表现对象全部的真实模型。在一些应用中，比如工程装配培训，当然此时虚拟对象应尽可能接近真实，以便能最大限度地提高训练效果。



图1-42 随机生成的包含100个顶点的多边形——你看到了什么？

然而，虚拟环境需要的不仅仅是对象的辨识——如在一张纸上的写意手法的勾画。虚拟环境中的对象构成参与者的环境：他或她所在的那个世界。此时我们要强调传感数据和本体感受之间匹配的重要性。关于这一点的一个好的例证是头部运动视差——当我转动头或者移动上身同时头部从一侧转到另一侧时，视觉信号应该依照平时对视差的体验而改变。如果真是这样的话，那么就会有很强的线索证明我的确位于这些对象当中。但是仍然存在另外的一个决定性问题：如果我能伸手并触摸到某个环境，如果我能穿行于其间并能呆在任何想呆的地方从而能接触各种不同的对象，那么我才是存在于一个环境中。举例来说，我站在一张桌子旁，我能移动到那张椅子处，而且可以弯下腰并伸出手拉那把椅子。那只手不一定非得是“我的”手，可以是一只跟踪真实手的虚拟手。但不管怎样，在这个将自己置身于对象中的幻觉中有同样的本体感受和传感数据的匹配。除了沉浸式虚拟环境以外没有什么其他技术手段能产生置身其中并能伸手接触虚拟对象的幻觉了。

这一点也与有关感知的终点有密切关系——Gregory 非常强调探索，强调建立在视觉基础
46 上的感知理解，建立在主动参与和对世界操作基础上的行动理解之间的密切关联。沉浸式虚拟环境的一个基本特征就在于它们能提供这种可能性——不是仅仅能看到周围的世界，而且能使用你的身体去探究和了解它。这一点很像在一个陌生城市中搭车旅行和亲自驾车旅行之间的差别。后一种情况与前一种情况相比，你会对路径有更深的了解和记忆——至少在这两种情况下你观察环境的方式将会是相当不同的。Held和Hein做了一个非常著名的实验（Gregory，1998b）说明了这一点：两只小猫在黑暗中长大，每一次当其中的一只积极地探究视觉环境的时候，另一只总是被动地跟随其后。两者对视觉学习有相同的机会，但是只有其中一只是主动的。实验表明只有那只积极地探究环境的猫学会了看。

反复实验显示，那些有机会全身投入虚拟环境的人较之那些只是在虚拟环境中看一看的人来说具有更强的存在感（Slater and Steed，2000）。沉浸式系统提供了这种独特的能力,允许人们积极地探究和了解一个环境。

让我们返回到本节开头所引述的Stark教授的一段话：“虚拟现实之所以能起作用是因为现实是虚拟的。”我们对世界的大多数视觉理解与我们早已存在的认知之间存在紧密联系，我们不必因为存在感觉而在VE中忠实地复制现实。这是个好消息，正如在这本书的后续各章中将要看到的，采用今天的计算机图形学技术在实时中制造视觉上的“真实”是件极端困难的事情。

1.7 小结

本章介绍了这本书的写作动机、内容范围以及写作方法。我们从抽象到具体，从真实到

实时。本章我们介绍了虚拟环境内容的概念，简短地描述了一个虚拟环境是如何由图形对象组成的。典型的虚拟场景用描述这些对象的数据库来表示，该数据库中包括了虚拟对象的几何属性、材质属性、反射属性和行为属性等方面的信息。

我们对感知问题给予了相当的关注，讨论了关于虚拟现实是如何工作的这样一些问题。我们主要是在用我们的想像力去看。我们每天生活的世界也是“虚幻世界”，而不是只有梦境和虚拟环境是“虚幻世界”。

既然我们在用我们的想像力去看，虚拟环境需要产生必要的信号，以便我们的视觉记忆去选择虚拟世界设计者想要的那种解释。能产生虚拟对象幻觉的环境要素的相关研究目前还非常少，我们这里把它作为虚拟环境的一个重要研究方向提出来。对此主题引起相当关注的应用领域是在大范围环境中路径求解方面——在这种环境中哪些关键特征是人们在求解路径 47
中最感兴趣的呢（Darken et al.，1999）？我们在有关存在感知的讨论中只是略微谈到这一点，但是对环境中的关键特征还需要很多深入的研究。

在下一章中我们将讨论一些基础数学知识。然后在第3章中，将在非常抽象的层次上讨论光照的问题。然后接着讨论人对光的反应问题——特别是颜色问题。这几章一并构成本书的介绍部分。

第2章 虚拟环境的数学基础

2.1 引言

在这一章中我们介绍一些必要的数学知识，它们在有关虚拟环境的描述和推理中都是至关重要的。基本上这些数学知识都是用来描述对象、操作对象的，同时也用于光照表示。

这里给出的讨论从数学角度看是不严格的，我们更多的是考虑了它的直观性，同时也不是面面俱到，所讲述的内容只是为了满足书中内容的需要。

2.2 维度

首先，我们要介绍一下空间维度的概念。它指的是对象在空间范围内运动的自由度数量。它也用来描述空间中位置所需要的最小的独立值个数。

如果能够想像出“对象”包含在一个零维的环境中的话，那么它们将是完全不能移动的。
48 而且，在零维空间中不需要任何信息来描述对象的位置：它只是“在那里”而已。一个零维空间只是一个点（根据这样的空间中对象的观点，在它的外面当然什么也没有）。我们知道数学点是没有“厚度”的，它没有面积，是一个“无穷小的”的点，在该点上的对象几乎可以看成一个点。

在一维空间里的对象能够向“左”或向“右”移动：它们只能在一个方向上来回运动。请注意这里“左”和“右”是用引号的，因为它们只是一个相对的概念，这个方向的规定是相当任意的。实际上很容易被误解，因为它们暗示水平的概念。然而，在一维空间中是不可能谈论“水平”这个概念的，因为它需要有一个大于一维的参考框架，只有在这样的参考框架内才能谈论清楚什么是水平。一维空间可以认为是无限长且连续的直线（对一维空间中的对象来说，其存在空间就是这条直线）。为了描述这种空间，我们可以构造一个坐标框架。选择任意一个点，标记为原点 0。从该点出发的一个方向我们称为正方向，相反的方向称为负方向。整个直线在正方向的那一部分叫做正半空间，同样地，在负方向的那一部分叫做负半空间。我们需要先选择某一种度量单位，这样，空间中任何点的位置就可以用一个前面带有“+”或“-”符号的一个数来表示该点到原点间的距离了。位于正半空间中的点被标记为“+”，位于负半空间中的点被标记为“-”。

在一维空间中可以存在无限多个“对象”。这些对象可能是零维的（例如单一点），或者是一维的（例如一段区间）。一般一维空间中的对象可以表现为点的集合。

这种空间可以有很多用处，举例来说，用来表示在一条高速公路上的车流。这里高速公路本身就表示那条“直线”，而车辆被表示为区间 $[a, b]$。随着车辆在高速公路上行进，表示它的区间$[a, b]$也要发生改变（虽然长度$b-a$保持常数）。即使是在这样简单的空间中仍然存在某些重要的计算问题。举例来说，设有一个对象集合$[a_1, b_1]$，$[a_2, b_2]$，…，$[a_n, b_n]$，快速找出所有相交的对象集合。或假设在一个交互式环境中，操作者通过一维光标选择某个对象。程序需要解决下面的问题：给定直线上的任一点 x，尽快地在一组正被指点的对象当中找到区间 $[a_i, b_i]$包含 x的那些对象。

二维空间中的对象既可以在“左”和“右”的方向上移动，也可以在“上”和“下”方向上移动（同样，这些都是任意标记的，只是一种相对的概念）。而且，通过在上面这些方向上的移动，对象所能到达的任一点实际上都能通过直线上的移动直接达到。为了要描述这种空间，我们需要两条彼此正交的（相互垂直）无限长直线来表示，它们相交于一点，该点称为原 49
点。这个交点是二维坐标系统的原点。通常，一个轴是“水平的”，我们习惯将它叫做X轴，而另一个是“垂直的”，习惯叫做Y轴。原点处坐标为(0，0)。在二维空间中的任一点可以用一对坐标值(x，y)来表示，这里x表示沿着X轴方向离原点的距离，y表示沿着Y轴方向离原点的距离。同样，在这种坐标系中（通常称为笛卡尔坐标系），需要规定一种度量单位来描述距离。设 $p_1=(x_1, y_1)$和 $p_2=(x_2, y_2)$是两个给定点，那么从p_1点到p_2点的距离是：

$$|p_1-p_2|=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2} \tag{2-1}$$

三维空间的描述是相似的。我们使用笛卡尔坐标系统，作为一个三维空间，我们选择某一个方便的点（任意的一个点）作为原点，同时选择三个相互正交的轴，分别称为X轴、Y轴和Z轴作为主坐标轴。所有的点用到原点的位移量来描述，分别用三个沿平行于主轴方向的位移量表示。注意原点的坐标是(0，0，0)，坐标系统是连续的。通常一个点表示为(x，y，z)。在三维空间中坐标轴的安排有两种方法，如图2-1所示。一种称为左手坐标系，另一种称为右手坐标系，我们可以任选其中之一。无论哪种情形，让拇指与食指保持垂直，同时中指保持与食指垂直，且让你的食指指向Y轴方向，拇指指向X轴方向，则中指所指的方向就是Z轴方向。

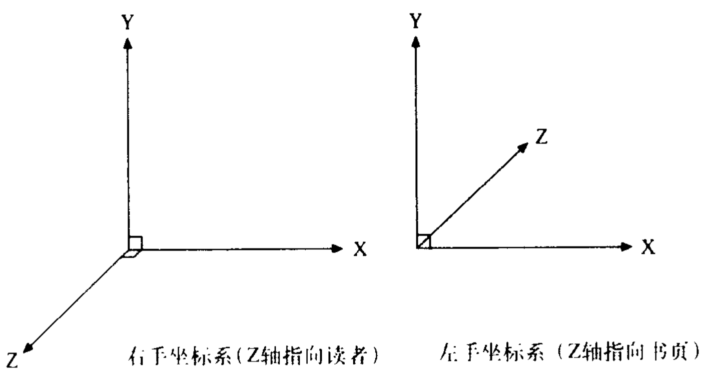


图2-1 右手坐标系和左手坐标系

2.3 位置和方向：点和矢量

点和矢量之间的关系

当我们确定了原点和主轴之后，点所表示的位置就是确定的了。矢量规定了一个方向，以及在该方向上的一个量，同时矢量也表示了两个点间的差值。在三维空间中，同样用(x, y, 50
z)表示一个矢量，根据上下文我们就能弄清楚它到底是表示一个点还是表示一个矢量。

假设（4，5，6）和（3，4，5）是两个点，那么（1，1，1）=（4，5，6）－（3，4，5）是一个矢量，表示方向（1，1，1）。矢量(x, y, z)可以被图示化为一端在原点，另一端在点(x,

y, z)且在(x, y, z)处带有箭头的一条直线，如图2-2所示。

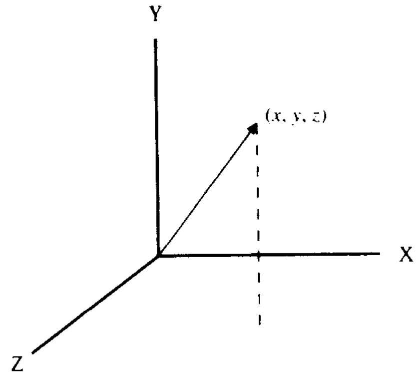


图2-2 三维中的矢量

矢量也表示两点间的差值：

$$vector = point_2 - point_1 \quad (2\text{-}2)$$

从这个公式我们也可以将一个点理解为点与矢量的和：

$$point_2 = point_1 + vector \quad (2\text{-}3)$$

矢量的加法

矢量可以做求和运算，遵循平行四边形规则（式2-4）。矢量求和的例子如图2-3。

51

$$\begin{aligned} v_1 &= (x_1, y_1, z_1) \\ v_2 &= (x_2, y_2, z_2) \\ v_1 + v_2 &= (x_1+x_2, y_1+y_2, z_1+z_2) \end{aligned} \quad (2\text{-}4)$$

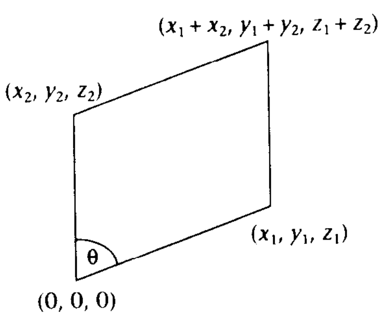


图2-3 矢量的加法

注意虽然把两个矢量相加是有意义的，但是两个点相加就没有任何意义了。（留给读者的问题：为什么？）

矢量的伸缩

设$v=(x, y, z)$是任意一个矢量，那么通过对矢量各分量乘以同一个实常数实现矢量的伸缩（例如放大）。设这个实常数为λ，则矢量伸缩可以用下面公式表示：

$$\lambda v=(\lambda x, \lambda y, \lambda z) \tag{2-5}$$

若$\lambda>0$，则所得的新矢量与原矢量方向相同，新矢量的长度为：

$$|\lambda v|=|\lambda||v|，这里|\lambda|=\lambda(如果\lambda>0)，|\lambda|=-\lambda(如果\lambda<0) \tag{2-6}$$

若有两个矢量v_1、v_2，另有两个实数λ_1、λ_2，则$\lambda_1 v_1+\lambda_2 v_2$也是一个矢量。

矢量的模

有三个特殊矢量构成整个空间的基：

$$\begin{aligned} e_1&=(1,0,0)\\ e_2&=(0,1,0)\\ e_3&=(0,0,1) \end{aligned} \tag{2-7}$$

也就是说任何矢量(x, y, z)可以表示成这三个矢量的线性组合，有：

$$(x, y, z)= xe_1+ ye_2+ze_3 \tag{2-8}$$

这三个主矢量通常也分别被称为i、j和k。

矢量的模指的是矢量的长度（事实上它提供了三维空间度量方法）。设$v=(x, y, z)$是一个矢量，那么它的模由$|v|$表示，定义为：

$$|v|=\sqrt{x^2+y^2+z^2} \tag{2-9}$$ 52

这是从原点到点(x, y, z)的距离。所以如果p_1和p_2是两个点，那么p_1-p_2是一个矢量，该两点之间的距离是p_1-p_2的模$|p_2-p_1|$。

给定任意矢量$v=(x, y, z)$，假若我们只对它的方向感兴趣，不在乎它的长度。那么对于许多目的，使用有相同方向但长度为1的另外一个矢量是很方便的。我们能通过对最初的矢量规范化求出这个单位矢量。如下所示：

$$\text{norm}(v)=\frac{v}{|v|} \tag{2-10}$$

显然有$|\text{norm}(v)|=1$。

两个矢量的内积（点乘）

同样设有两个矢量v_1和v_2。其点乘表示为$v_1 \cdot v_2$。它给我们提供了两矢量之间角度关系的重要信息。定义如下：

$$v_1 \cdot v_2= x_1x_2+ y_1y_2+z_1z_2 \tag{2-11}$$

将每个矢量v_1和v_2先规范化，然后计算两个规范化矢量的点乘。这给我们一个重要的结果：

$$\cos\theta=\frac{v_1 \cdot v_2}{|v_1|\times|v_2|} \tag{2-12}$$

这里θ表示两个矢量之间的角度。

因此点乘正比于两个矢量之间夹角的余弦。已知$\cos 0=1$和$\cos\frac{\pi}{2}=0$。

第一个结果的含义是单位长度的矢量与自身的点乘结果为1。第二个结果的含义是两个彼此垂直的矢量（我们称作矢量正交）的点乘为0。这个结果对以后各章十分有用。

矢量叉乘

关于矢量的另一个二元操作在几何上十分重要——即两个矢量的叉乘，标记为 $v_1 \times v_2$。它构成一个新的矢量，如下所示：

$$v_1 \times v_2=(y_1z_2-y_2z_1,\ x_2z_1-x_1z_2,\ x_1y_2-x_2y_1) \tag{2-13}$$

叉乘的性质有：第一，

$$|v_1 \times v_2| = |v_1||v_2|\sin\theta \tag{2-14}$$

这里θ是两矢量之间的角度。第二，也是在计算机图形学中具有重要作用的一个性质，矢量$v_1 \times v_2$ 垂直于v_1和v_2矢量。对此的另外一种说法是（尽管有点超前）v_1和v_2构成一个平面。它们的叉乘是该平面的法向（即该矢量是垂直于此平面的）。现在有一个
53 重要的问题是关于新矢量的方向的——如果v_1和v_2在你面前的书页平面上（如图2-4所示），那么新矢量会是指向你还是在书页平面之后远离你？回答是v_1、v_2以及$v_1 \times v_2$三矢量遵从右手坐标系：即如果你让拇指指向v_1方向，食指指向v_2方向，那么$v_1 \times v_2$将与你的中指方向相同。通常很容易看出$v_1 \times v_2= - v_2 \times v_1$（这种性质叫做反对称性，点乘具有对称性）。

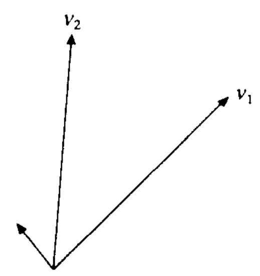


图2-4 叉乘的构成

式（2-13）总是不容易记住。有两个方法可以导出它。第一种方法是通过下列的行列式：

$$D=\begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} \tag{2-15}$$

这里i、j和 k是式（2-7）中的主单位矢量。

展开这个行列式：

$$\begin{aligned} D&=(iy_1z_2+jz_1x_2+kx_1y_2)-(kx_2y_1+iy_2z_1+jz_2x_1) \\ &=i(y_1z_2-y_2z_1)-j(z_2x_1-z_1x_2)+k(x_1y_2-x_2y_1) \\ &=(y_1z_2-y_2z_1,\ x_2z_1-x_1z_2,\ x_1y_2-x_2y_1) \\ &=v_1 \times v_2 \end{aligned} \tag{2-16}$$

还有另一种比较简单的导出式（2-13）的方法。考虑：

$$\begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix} \tag{2-17}$$

让我们遮住第一个行，计算行列式（叉乘）剩余的两行。得出叉乘的x坐标$y_1z_2-y_2z_1$。同样地，遮住行列式的第二行，计算剩余两行的值，得出$z_2x_1-z_1x_2$，再乘以－1就得出叉乘的y
54 坐标了。最后遮住第三行，计算剩余行列式的值$x_1y_2-x_2y_1$即是叉乘的z坐标了。当然，有些人会觉得式（2-13）很容易记住，而其他人会觉得记住公式的导出方法更容易。

关于主单位矢量 i、j和 k 之间一些关系对于今后很有用。如果我们将 $a \times b$写成 ab，将 aa 写成a^2，那么很容易证明下列各式：

$$i^2=j^2=k^2=(0,\ 0,\ 0)$$

$$
\begin{aligned}
ij &= k = -ji \\
jk &= i = -kj \\
ki &= j = -ik \\
(ij)k &= i(jk) = ijk = (0,\ 0,\ 0)
\end{aligned}
\tag{2-18}
$$

式（2-18）的最后一行说明了无论按哪个顺序运算叉乘的结果都是一样的，例如无论是先计算ij然后再与k运算，还是先进行jk运算然后再与i运算，结果都是一样的，所以ijk的定义是明确的。

2.4 方向和角度

在单位球体上的方向

一个矢量包含了一个量（矢量的模或长度）和一个方向。在计算机图形学中我们最为感兴趣的是矢量的方向。一个有趣的也是很重要的一点是用三个标量x、y和z来表示一个矢量有很多冗余信息。从式（2-10）我们应该清楚地看到事实上只有两个量是必须的：在三维空间中所有可能的方向所构成的空间事实上是一个二维空间！理由是当我们定义一个方向时，只需要考虑规范化矢量（因为矢量长度无关紧要）。对任意规范化矢量(x, y, z)，显然有：

$$x^2+y^2+z^2=1 \tag{2-19}$$

因此，如果我们知道x和y的值（假定），那么使用式（2-19）就能计算出z的值（不包括它的符号）。任何方向可以更直观地用一根直线来表示，此直线从单位球体的原点出发到单位球体表面上的某一点（球体用式（2-19）表示）。所以所有可能的方向与单位球体表面上的各点一一对应。但是球体表面是一个二维空间实体——因此说所有的三维空间中的可能方向都是二维的。

球坐标表示

三维空间中点的球坐标表示通常是很有用的，在下个小节中就会用到。在图2-5中，P是某一点(x, y, z)，从原点O到P的距离是 $r=\sqrt{x^2+y^2+z^2}$ 。Q是P点在 XY 平面上的垂直投影。 55
角β是X 轴和OQ之间的夹角，角α是Z轴和OP之间的夹角。

根据假设，角$\Delta OPQ=\alpha$，那么有$OQ=r\sin\alpha$。从而得出：

$$
\begin{aligned}
x &= r\sin\alpha\cos\beta \\
y &= r\sin\alpha\sin\beta \\
z &= r\cos\alpha
\end{aligned}
\tag{2-20}
$$

因此，由球坐标(r, α, β)，很容易使用这些公式计算出相应的笛卡尔坐标(x, y, z)。逆关系也可以从下面公式求得：

$$
\begin{aligned}
\alpha &= a\cos\frac{z}{r} \\
\beta &= a\tan\frac{y}{x}
\end{aligned}
\tag{2-21}
$$

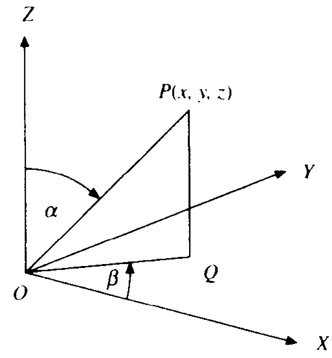


图2-5 球坐标

立体角

这启发了我们用另一种方式来表示一方向矢量：使用一组角度来定义单位球面上的一个点。图2-6给出了一个单位球体以及与其相交的矢量v，交点为P。从P点做到XY平面的垂线与平面相交于Q。矢量v的方向完全可以由两个角度$\phi=XOQ$和$\theta=ZOP$来确定。从这里我们可以得到一个重要的结果，那就是所有方向矢量的集合可以用如下所示的二维空间来表示：

$$\Omega=\left\{(\theta,\phi)\mid 0\leqslant \phi<2\pi,-\frac{\pi}{2}\leqslant \theta\leqslant \frac{\pi}{2}\right\} \tag{2-22}$$

在图2-6中我们有意给出了球面上围绕点P的小区域。假设该区域为A。相应于A的立体角被定义为：

$$\Gamma=\frac{A}{r^2} \tag{2-23}$$

56 这里r是球的半径。度量单位称为立体弧度。整个球体包含 $4\pi r^2/r^2=4\pi$立体弧度。立体角的计算类似于平常的角度，即等于它所围的圆周长度与圆的半径之比。

微分立体角是一个“微分区域”所对应的立体角。比如，在球体表面上一个面积趋于零的小面片就是一个微分区域。我们通常用dω来表示微分区域。这里将要给出根据方向或球面上的点所对应的角度θ和ϕ导出dω的公式，从中我们能够看到以上各概念之间的联系。

图2-7显示了半径为r的球面上的一个点 $p=(x, y, z)$以及在p处的微分立体角。假设p有球面坐标(θ, ϕ)。点p位于球面上高度为z圆心为C的圆周上。该圆的半径为$r\sin\theta$。因为沿着这个圆周的一个很小的水平扫略弧度一定是dϕ，因此有：

$$\mathrm{d}\phi=\frac{\mathrm{d}h}{r\sin\theta} \qquad \therefore \mathrm{d}h=r\,\mathrm{d}\phi\sin\theta \tag{2-24}$$

57

同样地，如果考虑在包含p点的大圆周上所扫略的角度dθ，则有：

$$\mathrm{d}\theta=\frac{\mathrm{d}v}{r} \qquad \therefore \mathrm{d}v=r\,\mathrm{d}\theta \tag{2-25}$$

因为立体角是面积除以半径的平方，从式（2-24）和式（2-25）我们可以得出所求的微分立体角是：

$$d\omega = \frac{dh \times dv}{r^2} = \sin\theta \, d\phi \, d\theta \quad (2\text{-}26)$$

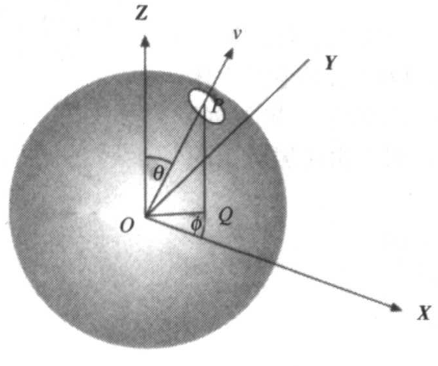


图2-6 用球面角度表示的方向矢量

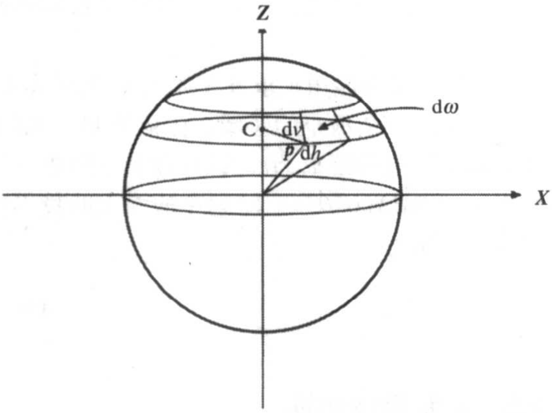


图2-7 微分立体角

投影区域

我们最后想介绍的一个概念是*投影区域*。在图2-8a中给出了一个区域A和一个平面P。考虑A在平面P上的投影——即A中的每个点沿着方向为v的光线投射并与P相交的交点所形成的图像。如果A与P是平行的，而且投影的方向垂直于A（如图中的 n，通常我们称之为A的*法向*），那么投影区域与A最初的区域完全相同。如果投影的方向不是垂直于A（例如图中v所示），或者区域和平面不是平行的（如图2-8b所示），那么投影区域将比最初的区域小。最重要的是投影方向和法向（垂直线）之间的*角度余弦*。角度越小，其余弦值就越接近于1，投影区域面积也接近于最初区域的面积。角度越大，投影区域面积就会变得越小，当角度接近π/2，投影区域将消失。注意余弦值是法向和投影方向的点乘：$\cos\theta = n \cdot v$，假定矢量已经是规范化了的。 58

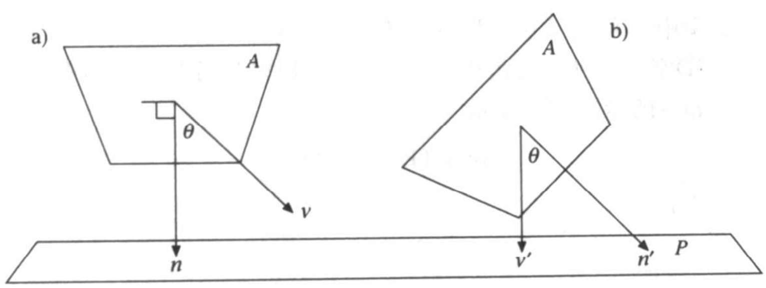


图2-8 投影区域

投影区域的定义是：

$$A^v = A\cos\theta$$

这里 v 是投影的方向

n 是A的法向

$$\cos\theta = n \cdot v \quad (2\text{-}27)$$

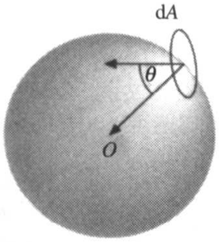


图2-9 微分区域所对应的立体角

为避免混乱，这里A不是面积的概念，而只是表示一个区域。因此A表示最初的区域，而 $A^v= A\cos\theta$有时用来表示投影区域。如果A的实际面积用 $|A|$表示，那么$|A^v|=|A|\cos\theta$。然而，为方便起见通常不使用|…|符号。从上下文中我们应该清楚哪个表示面积，哪个表示面积所对应的区域。

投影区域也能用来求微分区域（“趋于零的区域”）的立体角。图2-9给出了任意一个很小的区域，标记为dA，我们希望从O点计算它的立体角，dA到O点的距离为r。以O为中心构造一个半径为r的球。朝向O的投影区域为d$A\cos\theta$——即它在球面上的投影区域。那么由区域包围的微分立体角为:

$$d\omega = \frac{dA\cos\theta}{r^2} \tag{2-28}$$

2.5 平面保持变换

重心组合：平面

59 在这一小节中我们要学习保持“平面”的变换。它在 3D 计算机图形学中具有重要意义，因为在3D计算机图形学中基本图元是平面多边形。当变换一个用多边形构造的对象时，我们需要这种变换一定要保证多边形平面的不变性。同时，这种变换对计算推理也是非常有用的——因为我们可以只转换多边形的顶点，并将转换后的顶点作为新多边形的顶点。如果不是这样的话，为了要得到转换后对象的形状，我们就不得不转换原多边形内的“每个点”（顶点、边界以及所有内部点）。首先来定义“平面”的概念，其次要定义平面保持变换的性质，通常我们称之为*仿射变换*。

线段

在计算机图形学中一种最基本的形状是连接两点的直线。一种最好的表示方法是它的*参数化形式*：给定在3D空间的两个点p_1和p_2（实际上可以是任意维数空间中的点），那么通过该两点的直线可以被如下所示的方程表示:

$$p(t)=(1-t)p_1+tp_2 \tag{2-29}$$

t为任何实数（见图2-10）。

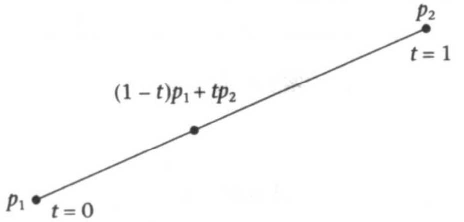


图2-10 线段的参数化表示

如果我们想要连结p_1到p_2的线段，那么t的取值范围为[0, 1]。这个结果将会在本书中多处

使用（其他形式的直线表示将会在第17章中讨论）。注意这个方程也可以写成如下形式：

$$p(t)=p_1+t(p_2-p_1) \tag{2-30}$$

这个形式的直线表示说明直线段可以看成是一个初始点p_1与一个矢量p_2-p_1之和，伸缩因子$t\in[0, 1]$。

重心组合

参数化直线表示是一种被称为重心组合的特殊情况。所谓重心组合指的是点的加权和，权值总和为1。设p_1，p_2，…，p_n是一个点的序列，又a_1，a_2，…，a_n是任一个实数序列，其和为1，那么 60

$$p=\sum_{i=1}^{n}\alpha_i p_i$$

(2-31)

$$且\quad \sum_{i=1}^{n}\alpha_i=1$$

定义了一个重心组合。显然当n=2时它就是式（2-29）中给出的直线。

暂时假设点p_1、p_2、…、p_n全部在同一个平面（P）上（第8.2节）。

假如我们让α_i在总和为1的约束条件下任意改变。每一组值（α_1，α_2，…，α_n）对应空间中的一个点p。考虑所有这些点（当然有无限多个）的集合，可以证明所得的面就是平面P，该平面通过p_1，p_2、…、p_n每个点。直观上是相当容易明白的。如果选择其中的任何两点，比如说p_a和p_b，我们可以设定除了α_a和α_b外所有的α_i=0，那么约束要求$\alpha_a+\alpha_b=1$，于是表示了P平面上过p_a和p_b的直线上的所有点。又任意选择该直线上的某一点，设它为p_{ab}。再选择另外一个初始点，设为p_c，则使用相同的参数我们所得到过p_{ab}和p_c直线的所有点都一定是在平面P上。进一步可以得出结论，即过p_a、p_b和p_c的平面上的所有点都属于P。用相同的参数和初始集合中的每个点，所得的那些点都一定位于相同的平面上，这个平面就是P平面。

凸多边形是指其所有内角都小于180度的多边形。凸多边形的另外一个性质是连接多边形的边界上两点的任何直线段完全包含在该多边形内部。如果我们进一步限制权值是非负的，$\alpha_i\geqslant 0$，i=1、…，n，那么此时很容易发现P变成了包含所有顶点p_1，p_2，…，p_n的一个最小凸多边形（包括内部在内）。当以这些顶点为顶点的多边形为凸多边形时，实际上P就是该多边形。

更一般的情形，我们假设p_i不全部位于同一个平面上（当n=4时往往是这样）。如果我们限制α_i是非负的，那么随着α_i的变化式（2-31）所求得p都位于一个凸包内，该凸包包围所有的p_i点。这个凸包是包含p_i的最小多面体。数学上对凸包的描述是：与该凸包边界相交于两个点的线段完全位于凸包的内部或边界上。特别地，这些顶点的质心（或“平均值”）将位于凸包之内，该平均值是：

$$\bar{p}=\frac{1}{n}\sum_{i=1}^{n}p_i \tag{2-32}$$ 61

仿射变换

仿射变换是一种保持重心组合的变换——因此也是平面保持变换。这个性质可以被精确

定义如下。设在3D空间中的p点是式（2-31）中的重心组合，假设f为从3D空间到3D空间的一个映射，$f(p)$仍然是这个空间中的一个点。那么f是仿射的，当且仅当下式成立：

$$f(p)=\sum_{i=1}^{n}\alpha_i f(p_i) \tag{2-33}$$

换句话说，仿射变换是这样的一种变换，当我们先计算重心组合点p然后计算相应的变换点，所得的结果与我们先求所有的点的变换$f(p_i)$，然后再找重心组合是一样的。从计算上讲，变换由重心组合所构成的对象时，后一种方式总是比前一种方式要高效得多。当我们考虑平面情形时，很显然平面多边形的仿射变换可以先转换它的每一个顶点，然后从这些新顶点来构造新的多边形。

仿射变换的矩阵表示

式（2-33）给出了仿射变换的性质，但是它并不是一种构造性的公式。变换所采取的形式是什么？我们在这一小节中将导出这种变换的形式。

因为p是3D空间中的一个点，它可表示为坐标形式：(x_1, x_2, x_3)。另设三个主单位矢量：

$e_1=(1, 0, 0)$

$e_2=(0, 1, 0)$

$e_3=(0, 0, 1)$

定义 $e_4=(0, 0, 0)$。则有：

$$\begin{aligned}p&=x_1e_1+x_2e_2+x_3e_3+x_4e_4\\&=\sum_{i=1}^{4}x_ie_i\end{aligned} \tag{2-34}$$

这里x_4可以是任何值。既然x_4可以任意指定，它都不会影响式（2-34），我们将选择

$$x_4=1-x_1-x_2-x_3$$

所以有：

62

$$\sum_{i=1}^{4}x_i=1 \tag{2-35}$$

现在考虑主单位矢量e_i。如前所述，设f是一个仿射变换，那么$f(e_i)$在3D空间中，因此一定有坐标λ_{ij}满足：

$$\begin{aligned}f(e_i)&=(\lambda_{i1},\lambda_{i2},\lambda_{i3})\\&=\sum_{j=1}^{3}\lambda_{ij}e_j\end{aligned} \tag{2-36}$$

从式（2-34）和式（2-35）我们得到：

$$f(p)=\sum_{i=1}^{4}x_i f(e_i) \tag{2-37}$$

将式（2-36）代入式（2-37）得：

$$f(p)=\sum_{i=1}^{4}x_i\sum_{j=1}^{3}\lambda_{ij}e_j \tag{2-38}$$

改变式（2-38）的求和次序得：

$$f(p)=\sum_{j=1}^{3}e_j\sum_{i=1}^{4}x_i\lambda_{ij} \tag{2-39}$$

已知$x_4=1-x_1-x_2-x_3$，令$\mu_{ij}=\lambda_{ij}-\lambda_{4j}$，代入上式得：

$$f(p)=\sum_{j=1}^{3}e_j\left(\sum_{i=1}^{3}x_i\mu_{ij}+\lambda_{4j}\right) \tag{2-40}$$

由e_j的定义，我们将$f(p)$展成坐标形式：

$$f(p)=\left(\sum_{i=1}^{3}x_i\mu_{i1}+\lambda_{41},\sum_{i=1}^{3}x_i\mu_{i2}+\lambda_{42},\sum_{i=1}^{3}x_i\mu_{i3}+\lambda_{43}\right) \tag{2-41}$$

我们将它写成更一般的形式，现在假设$p=(x,\ y,\ z)$，$f(p)=(x',\ y',\ z')$。如果f是一个仿射变换，那么每个新坐标可以由原坐标简单的仿射组合表示，即：

$$\begin{aligned}x'&=a_{11}x+a_{21}y+a_{31}z+a_{41}\\y'&=a_{12}x+a_{22}y+a_{32}z+a_{42}\\z'&=a_{13}x+a_{23}y+a_{33}z+a_{43}\end{aligned} \tag{2-42}$$

a_{ij}为一些常数。 63

矩阵表示

式（2-42）给出了从点(x,y,z)到点（x'，y'，z'）仿射变换的一般形式。式（2-42）用矩阵形式表示是很方便的。如前所述，假设$p=(x,y,z)$，$f(p)=(x',y',z')$，这里f是仿射的平面保持变换。该变换有三种矩阵表示方法。第一种方法是

$$f(x,y,z)=(x',y',z')=(x,y,z)\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\end{bmatrix}+\begin{bmatrix}a_{41}\\a_{42}\\a_{43}\end{bmatrix} \tag{2-43}$$

我们也可以去掉平移矢量这一项，写成如下形式：

$$(x',y',z')=(x,y,z,1)\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\\a_{41}&a_{42}&a_{43}\end{bmatrix} \tag{2-44}$$

为了要计算连续的一组变换，我们做矩阵的乘法运算。利用式（2-43）和式（2-44）的形式是不可能的，因为递次矩阵不适合做乘法运算。为此我们要介绍3D空间点的一种齐次表示形式：$(x,\ y,\ z)\equiv(x,\ y,\ z,\ 1)$，由此仿射变换得到一种齐次矩阵表示形式如下：

$$(x',y',z',1)=(x,y,z,1)\begin{bmatrix}a_{11}&a_{12}&a_{13}&0\\a_{21}&a_{22}&a_{23}&0\\a_{31}&a_{32}&a_{33}&0\\a_{41}&a_{42}&a_{43}&1\end{bmatrix} \tag{2-45}$$

现在所有的表示矩阵都是4×4的，因而能够彼此相乘。

通常对任何三维空间点$(x,\ y,\ z)$都有无穷多个$(wx,\ wy,\ wz,\ w)$这样的等价表示（对任何$w\neq0$），这可以看成是四维空间中的一条直线的参数化表示，该直线通过原点和点$(x,\ y,$

z，1)，w为参数。对任何这种齐次点，其在三维空间中的对应点可以通过用w除以一致表示的各项来得到。事实上，整个三维空间可以被看成四维空间的一个三维子空间，该三维子空间由超平面w=1定义。因此给定任何一个齐次点（X，Y，Z，W）（$W\neq 0$），其等价的三维点是（X/W、Y/W、Z/W）。三维空间点的传统表示(x，y，z)在齐次空间中就是(x，y，z，1)（w=1）。

标准变换

在这一小节中我们学习有关在各个主轴方向上的平移、伸缩和旋转的标准变换。

平移矩阵。当平移量表示为矢量（a，b，c）时，我们可以用如下矩阵来表示这样一个变换：

64

$$T(a,b,c)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\a&b&c&1\end{bmatrix} \tag{2-46}$$

因为有：(x, y, z, 1) T(a, b, c)=(x+a, y+b, z+c)。

伸缩矩阵。对x、y和z相对于原点分别伸缩a、b和c的变换矩阵为：

$$S(a,b,c)=\begin{bmatrix}a&0&0&0\\0&b&0&0\\0&0&c&0\\0&0&0&1\end{bmatrix} \tag{2-47}$$

因为有：(x, y, z,1)S (a, b, c)=(ax, by, cz)。

旋转矩阵。旋转只能是相对于一个轴做旋转——绕某个点的旋转没有定义，这是因为绕一个点旋转某个角度θ的可能路径有无穷多个。绕一个轴所做的旋转只有两条可能的路径，分别对应两个所选择的方向。我们定义一个方向为正向，即当我们顺着该轴看向原点的时候，反时针的那个方向为正向（假定是右手坐标系，参见第1章）。这里只考虑关于三个主轴X轴、Y轴和Z轴的旋转情况。

绕Z轴的旋转。当我们绕Z轴去旋转一个点(x, y, z)的时候，z坐标保持不变，因为此时该点的运动轨迹是一个圆，

65 该圆中心在z轴上，距离原点为z。因此，我们可以不去管z坐标，只考虑x和y的变化。这里绕Z轴旋转点(x, y, z)，旋转角度为正向（反时针方向）（如图2-11）。变换是(x, y, z,1) $R_z(\theta)$=(x', y', z)。

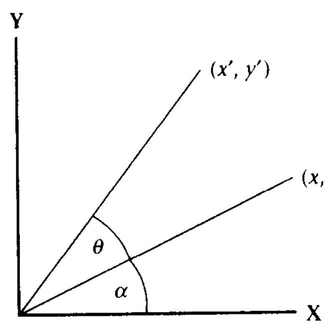


图2-11　极坐标——点的旋转

显然有下列结果：

$$\begin{aligned}x'&=r\cos(\alpha+\theta)=r\cos\alpha\cos\theta-r\sin\alpha\sin\theta=x\cos\theta-y\sin\theta\\y'&=r\sin(\alpha+\theta)=r\sin\alpha\cos\theta+r\sin\theta\sin\alpha=x\sin\theta+y\cos\theta\\z'&=z\end{aligned} \tag{2-48}$$

这里r是从原点到(x, y)的距离。

这可以表示成矩阵形式如下：

$$R_z(\theta)=\begin{bmatrix}\cos\theta & \sin\theta & 0 & 0\\ -\sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2-49}$$

绕Y轴的旋转。根据对称性，用z 替换x，同时用x替换y，则得到绕Y轴旋转的矩阵表示为：

$$R_y(\theta)=\begin{bmatrix}\cos\theta & 0 & -\sin\theta & 0\\ 0 & 1 & 0 & 0\\ \sin\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2-50}$$

绕X轴的旋转。同样根据对称性，绕X轴的旋转矩阵为：

$$R_x(\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & \sin\theta & 0\\ 0 & -\sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2-51}$$

逆变换。每一个变换都有一个逆变换：

$$\begin{aligned} &T^{-1}(a,b,c)=T(-a,-b,-c)\\ &S^{-1}(a,b,c)=S\left(\frac{1}{a},\frac{1}{b},\frac{1}{c}\right)\quad a\neq 0,b\neq 0,c\neq 0\\ &R_s^{-1}(\theta)=R_s(-\theta)\qquad \text{对任何 } s=x,\ y,\ z\end{aligned} \tag{2-52}$$ 66

变换的合成。假设我们从一点$p=(x, y, z)\equiv(x, y, z, 1)$开始，如果应用变换$M_0$到这一点，我们得到$p_1=pM_0$。如果应用变换$M_1$，则我们得到的是点 $p_2= p_1M_1= PM_0M_1$。如果以这样的方式连续应用变换M_3、M_4、…很容易看到所得到的点的一般表示为 $p_i=pM_0M_1\cdots M_i$。因此在一个点上的变换序列可以用相应的矩阵乘法表示，矩阵乘法的顺序与对应的变换顺序是一致的。从效率的角度看，尤其当我们需要以相同的变换序列作用到很多点时，先求出矩阵$M=M_0M_1\cdots M_i$，然后用矩阵M作用到所有的点上。很清楚这样的变换是不可交换的，因为它们是矩阵乘法。即通常有$M_iM_j\neq M_jM_i$。顺序问题：伸缩一个对象然后平移它与首先平移它然后再伸缩所产生的结果是不一样的。

关于一个点 $q=(x, y, z)\neq(0, 0, 0)$伸缩一个对象时，我们应该首先用一个平移变换将q点变成原点，然后再去伸缩，最后再做一个反向平移变换将q点变回到原处。因此关于某一个任意点(x, y, z)的伸缩可表示为：

$$T(-x,-y,-z)\,S(a, b, c)\,T(x, y, z) \tag{2-53}$$

绕任意轴的旋转是个较复杂的问题，将在下面讨论。

绕任意轴的旋转。设旋转矩阵 R_X、R_Y和R_Z是绕各主轴旋转的矩阵。仅有这些旋转是不够的——举例来说，除非一个对象中心碰巧在某个主轴上，否则我们就无法绕一个穿过该对象的轴线旋转它。而这种旋转又是非常重要的——通常为了更好地理解一个对象，我们需要绕穿过它的一个轴旋转，或至少是绕一个接近它的轴旋转。

在这一小节中我们构造一个合成变换矩阵(R)来表示绕过指定的两点 $p_1=(x_1, y_1, z_1)$，$p_2=(x_2,$

y_2, z_2)的轴线旋转角度α。矩阵R的构造过程如下：

（1）将p_1平移到原点处：$T(-p_1)$;

（2）设$(x, y, z)=p_2-p_1$是线段的另一端，以球坐标的形式表示为（r,ϕ,θ）。（这里$\alpha=\phi$, $\beta=\theta$, 参阅图2-5）；

（3）绕Z轴旋转$-\theta$，将Q置于ZX平面上：$R_z(-\theta)$;

（4）绕Y轴旋转$-\phi$，这样OP与Z轴重合：$R_y(-\phi)$。

通过对这些过程的组合，我们定义旋转矩阵$M=T(-p_1)R_z(-\theta)R_y(-\phi)$，该变换使得旋转轴线成为Z轴。现在开始做旋转变换$R_z(\alpha)$，然后应用逆变换$M^{-1}$。因此完整的变换可以表示为：

$$R=MR_z(\alpha)M^{-1} \tag{2-54}$$

这里：

$$\begin{aligned} M &= T(-p_1)R_z(-\theta)R_y(-\phi) \\ M^{-1} &= R_y(\phi)R_z(\theta)T(p_1) \end{aligned} \tag{2-55}$$

2.6 四元数

定义

四元数是复数思想的一种推广。它尤其在计算机图形学中描述旋转时特别有用，也对创
67 建插值旋转的关键帧动画序列有用。这里我们将对四元数给出粗略定义和简单应用，不进行详细、严格的数学描述和分析。首先我们将定义什么是四元数以及关于它们的一些运算，然后将给出关于旋转的几何解释。

假如u_0是一个标量值，$\boldsymbol{u}=(u_1, u_2, u_3)$是一个矢量，那么四元组$(u_1u_0, u_1, u_2, u_3)$代表一个四元数，定义为：

$$\begin{gathered} u=u_0+\boldsymbol{u} \\ \text{或者是} \\ u=u_0+\boldsymbol{i}u_1+\boldsymbol{j}u_2+\boldsymbol{k}u_3 \end{gathered} \tag{2-56}$$

这里对$\boldsymbol{i}$、$\boldsymbol{j}$和$\boldsymbol{k}$的定义如下：

$$\begin{aligned} \boldsymbol{i}^2 &= \boldsymbol{j}^2=\boldsymbol{k}^2=\boldsymbol{ijk}=-1 \\ \boldsymbol{ij} &= \boldsymbol{k}=-\boldsymbol{ji} \\ \boldsymbol{jk} &= \boldsymbol{i}=-\boldsymbol{kj} \\ \boldsymbol{ki} &= \boldsymbol{j}=-\boldsymbol{ik} \end{aligned} \tag{2-57}$$

式（2-57）第一行与虚数的定义i($i^2=-1$)相同。其他行与式（2-58）相同，该公式为主单位矢量间叉乘的定义。所有这些解释对四元数都是有效的。

四元数u的共轭是：

$$u^*=u_0-\boldsymbol{u} \tag{2-58}$$

纯四元数是指它的标量部分$u_0=0$。显然纯四元数与在3D空间中的矢量存在一致关系。

使用这些定义的目的是为了要给出四元数运算的定义。

四元数加法

假设 $v=(v_0, v_1, v_2, v_3)$是一个四元数，那么

$$\begin{aligned} u+v &= (u_0+v_0)+\boldsymbol{i}(u_1+v_1)+\boldsymbol{j}(u_2+v_2)+\boldsymbol{k}(u_3+v_3) \\ &= (u_0+v_0,\ u_1+v_1,\ u_2+v_2,\ u_3+v_3) \end{aligned} \tag{2-59}$$

注意加法是可交换的（$u+v=v+u$）。四元数的零元0（所有元素皆为0）对任何u有$u+0=0+u$。

标量乘法

68

如果u是一个四元数，c为一个标量，那么

$$\begin{aligned} c\cdot u &= c(u_0+\boldsymbol{i}u_1+\boldsymbol{j}u_2+\boldsymbol{k}u_3) \\ &= (cu_0, cu_1, cu_2, cu_3) \end{aligned} \tag{2-60}$$

四元数乘法

对$(u_0+\boldsymbol{i}u_1+\boldsymbol{j}u_2+\boldsymbol{k}u_3)\times(v_0+\boldsymbol{i}v_1+\boldsymbol{j}v_2+\boldsymbol{k}v_3)$按一般方式执行乘法展开，但是对$a\times b$各个项的解释是不一样的：当$a$、$b$都是标量时如平常的乘法；当它们为不同的矢量时如矢量叉乘；当一个为矢量而另一个为标量时如标量矢量乘法。然而在展开式中一定要保持矢量叉乘的顺序。

那么

$$\begin{aligned} u\times v = {} & u_0v_0+\boldsymbol{i}u_0v_1+\boldsymbol{j}u_0v_2+\boldsymbol{k}u_0v_3+ \\ & \boldsymbol{i}u_1v_0+\boldsymbol{i}^2u_1v_1+\boldsymbol{ij}u_1v_2+\boldsymbol{ik}u_1v_3+ \\ & \boldsymbol{j}u_2v_0+\boldsymbol{ji}u_2v_1+\boldsymbol{j}^2u_2v_2+\boldsymbol{jk}u_2v_3+ \\ & \boldsymbol{k}u_3v_0+\boldsymbol{ki}u_3v_1+\boldsymbol{kj}u_3v_2+\boldsymbol{k}^2u_3v_3 \end{aligned} \tag{2-61}$$

将相似项合并，并利用式（2-57）得：

$$\begin{aligned} u\times v = {} & u_0v_0-u_1v_1-u_2v_2-u_3v_3+ \\ & \boldsymbol{i}(u_0v_1+u_1v_0+u_2v_3-u_3v_2)+ \\ & \boldsymbol{j}(u_0v_2-u_1v_3+u_2v_0+u_3v_1)+ \\ & \boldsymbol{k}(u_0v_3+u_1v_2-u_2v_1+u_3v_0) \end{aligned} \tag{2-62}$$

再利用式(2-8)、式(2-11)和式(2-13)，我们有：

$$u\times v=[u_0v_0-(\boldsymbol{u}\cdot\boldsymbol{v})]+(\boldsymbol{u}\times\boldsymbol{v})+u_0\boldsymbol{v}+v_0\boldsymbol{u} \tag{2-63}$$

在方括号中为标量，表达式的余下部分是一个矢量。因此这个运算得到的是与式（2-56）相同的四元数形式。式（2-63）是四元数乘法的定义，显然四元数乘法得到的结果仍然是一个四元数。注意运算是不可交换的：$u\times v\neq v\times u$，但是它是可结合的：$u\times(v\times w)=(u\times v)\times w=u\times v\times w$。

假设u是一个纯四元数，$u=\boldsymbol{i}u_1+\boldsymbol{j}u_2+\boldsymbol{k}u_3$，那么

$$u^2=-(\boldsymbol{u}\cdot\boldsymbol{u}) \tag{2-64}$$

四元数的逆

考虑四元数的乘法和它的共轭：

69

$$u \times u^* = u_0^2 + u_1^2 + u_2^2 + u_3^2 \tag{2-65}$$

与向量类似，四元数的模|u|定义为：

$$|u|^2 = u \times u^* = u_0^2 + u_1^2 + u_2^2 + u_3^2 \tag{2-66}$$

单位四元数满足 |u|=1。根据式（2-64），我们推得任何纯四元数（I）有性质：

$$I^2 = -1 \tag{2-67}$$

对任何四元数的规范化就是用四元数的模去除它本身。因此，如果u是任意一个四元数，我们定义它的规范化为：

$$\text{norm}(u) = \frac{u}{|u|} \tag{2-68}$$

考虑到有：

$$\begin{aligned} u \times \text{norm}(u^*) &= \frac{u \times u^*}{|u|} \\ &= |u| \end{aligned} \tag{2-69}$$

由此可见可以定义四元数的逆为：

$$u^{-1} = \frac{u^*}{|u|^2} \tag{2-70}$$

因为有$u \times u^{-1} = u^{-1} \times u = 1$。

四元数的极坐标表示

设u为一个单位四元数，$u=u_0+\boldsymbol{u}$。因为u 是单位四元数，所以有：

$$\begin{aligned} |u|^2 &= u_0^2 + |\boldsymbol{u}|^2 = 1 \\ &= \cos^2\theta + \sin^2\theta \end{aligned} \tag{2-71}$$

对任意θ有 $-\pi<\theta\leqslant\pi$，我们总可以写成如下形式：

$$\begin{aligned} \boldsymbol{s} &= \frac{\boldsymbol{u}}{|\boldsymbol{u}|} \\ \therefore \boldsymbol{u} &= |\boldsymbol{u}|\boldsymbol{s} \end{aligned} \tag{2-72}$$

这里$\boldsymbol{s}$为一个单位矢量。因此我们有四元数的另外一种形式：

$$u=\cos\theta+\boldsymbol{s}\sin\theta \tag{2-73}$$

可以从另外一个角度来看待它。从式（2-72）得：

70

$$\boldsymbol{s} = \frac{1}{|\boldsymbol{u}|}(\boldsymbol{i}u_1 + \boldsymbol{j}u_2 + \boldsymbol{k}u_3) \tag{2-74}$$

那么

$$\boldsymbol{s} \times \boldsymbol{s} = \boldsymbol{s}^2 = -1 \tag{2-75}$$

因此有

$$\begin{aligned} u &= u_0 + \boldsymbol{u} \\ &= u_0 + |\boldsymbol{u}|\boldsymbol{s} \end{aligned} \tag{2-76}$$

这里$s^2=-1$，而且因为假定u是单位四元数，有 $u_0^2+|\boldsymbol{u}|^2=1$。这在结构上与复数$x+iy$的一般形式是一致的。这里$i^2=-1$ 是所谓的"虚"数。从复数理论我们知道 $x+iy=\cos\theta+i\sin\theta$（此时有$x^2+y^2=1$）。

因此任何单位四元数可以被分解为两个部分，如式（2-73）。这意味着四元数与角度（θ）和单位矢量（s）相关。现在我们就用这一点来说明四元数是如何用来计算旋转的。

四元数的旋转

我们先前看到了在纯四元数集合与三维空间矢量集合之间的一个对等关系。假设$\boldsymbol{p}$是一个矢量，p为对应的四元数。为了要使这一点完全清楚，依据式（2-56）定义有$p=0+\boldsymbol{p}$。设u是任意的单位四元数，有$u=\cos\theta+\boldsymbol{s}\sin\theta$。那行乘运算$upu^*$得到一个纯四元数$q=0+\boldsymbol{q}$，$\boldsymbol{q}$是绕轴$\boldsymbol{s}$旋转一个$2\theta$角度后的矢量。回想一下式（2-54），我们说明了如何通过矩阵变换完成点绕任意轴的旋转，因而通过四元数我们可以以十分简洁的方式完全达到同样的结果。

为了说明这个结果，我们首先考虑一个简单的例子——关于Z轴旋转$\boldsymbol{p}=(x, y, z)$（即$\boldsymbol{s}=(0, 0, 1)$）。首先让我们来计算up。

$$up=-(\boldsymbol{u}\cdot\boldsymbol{p})+\boldsymbol{u}\times\boldsymbol{p} \tag{2-77}$$

以上根据式（2-63）。但$\boldsymbol{u}=\boldsymbol{s}\sin\theta=(0, 0, \sin\theta)$，

$$\begin{aligned} up &= -z\sin\theta+(-y\sin\theta, x\sin\theta, 0)+\cos\theta(x,y,z) \\ &= -z\sin\theta+(x\cos\theta-y\sin\theta, x\sin\theta+y\cos\theta, z\cos\theta) \end{aligned} \tag{2-78}$$

现在我们用$u^*=0-(0, 0, \sin\theta)$右乘up，

$$upu^*=(x\cos 2\theta-y\sin 2\theta, x\sin 2\theta+y\cos 2\theta, z) \tag{2-79}$$

再一次得到式（2-48）的结果。在这个例子中，我们已经利用到了恒等式： 71

$$\begin{aligned} \cos 2\theta &= \cos^2\theta-\sin^2\theta \\ \sin 2\theta &= 2\sin\theta\cos\theta \end{aligned} \tag{2-80}$$

假设有一个旋转序列，其中各个旋转分别为绕轴 $\boldsymbol{s}_i$，$i=1$，…，n，角度为$2\theta_i$，那么可以通过应用以下序列中的四元数旋转算子得到：

$$u_n u_{n-1}\cdots u_1 p u_1^*\cdots u_{n-1}^* u_n^* \tag{2-81}$$

这里 $u_i=\cos\theta_i+\boldsymbol{s}_i\sin\theta_i$。这等价于求如下的四元数：

$$u=u_n u_{n-1}\cdots u_1 \tag{2-82}$$

然后把它当作一个旋转算子upu^*。典型情形是同时会有许多点需要转换。先计算出四元数u将极大地提高计算效率。一旦我们获得了四元数，就可以将它重复地应用到每个点上，而不用每一次都重新计算所有的中间旋转结果（除非像在动画应用中那样需要这些中间结果）。该结果也说明对于任意一个旋转序列来说，总是能被表示为一个单一旋转算子。

2.7 小结

本章从维度概念和如何使用直角坐标系来描述空间开始。我们用点来表示位置，同时用矢量来表示方向。我们介绍了立体角和投影区域的概念，以及它们之间的关系。最后研究了基于矩阵的仿射变换的概念，以及基于四元数的旋转的另外一种表示方法。

若是希望更多地了解计算机图形学所用到的数学内容，推荐给读者以下参考书目。第一本是 Andrew Glassner的有重要影响的参考书（Glassner，1995），该书包含了一个极为丰富的附录，覆盖了线性代数和图形学所用到的一些重要数学内容。第二本书为 Hoggar所著（Hoggar，1992），覆盖了数学的一个很大范围，包括线性代数、四元数以及不规则形状和混沌系统。最后要介绍的是Kuipers的著作，该书（Kuipers，1999）对旋转矩阵和四元数有一个完整的介绍。

我们花费了一些时间在点和矢量等基本概念上，目的是要提醒读者这些基本概念以及操作是非常基础的，对于计算机图形学的学习来说是必须掌握好的。本书大部分内容是反映计
72 算机图形学方面的实践，其中用来表示虚拟对象的几何类型是平面内的简单多边形类型（通常为三角形）。这种平面多边形从数学和计算的角度来讲都是很容易处理的。其优势在于绝大多数的形状可以用多边形或三角形集合近似表示，甚至那些弯曲形状都可以由它们来表示，假设有足够多的三角形来描述对象。虚拟世界中的“幻境”通常都是由多边形组成的。

第3章 光照——光亮度方程

3.1 光照：计算机图形学的基本问题

在第1章中我们介绍了场景的概念，所谓场景是一个图形对象的集合。用计算术语说它就是一个表现对象集合的数据结构。举例来说，每个对象本身就是多边形的一个集合。每个多边形是平面上点的一个序列。我们将在第8章中对这个问题给出详细讨论。

然而，“真实世界”是一个能“产生能量”的世界。我们的场景至今只不过是幻觉中的场景，因为它只是对一组形状的描述，并不存在真正的物质。能量一定要产生，场景一定要点亮，即使是由虚拟的光照亮的。计算机图形学关心的是场景虚拟模型的构造。对场景建模的工作相对来讲是个简单问题，尽管它很耗时。相比而言，场景的光照问题是计算机图形学中 73 的一个重要的核心概念和实际问题。这个问题是一个在场景中模拟光照的问题，这个过程中计算并不是永无休止地进行。作为结果的二维投影图像也应该看上去像是真的。事实上，让我们把这个问题变得更有趣和更具挑战性：我们不只是需要计算更快速，而且想要它达到实时。图像帧（即在场景里拍摄的虚拟相片）一定要快速地生成，以保证能赶上人在场景周围漫游时视线的变化（头部和眼睛的移动）——这样人们就能产生与在相应的真实场景中相同的视觉体验。理想的光照仿真应该是非常精确的，计算速度相当快，以致于人们不能区分出真实和虚拟。甚至在场景中的对象是可以移动的，光可以改变（例如日落），而且对在场景中的人的表示效果也应该作为光照计算的一部分。换句话说，我应该能从你的虚拟眼中看到我的虚拟影子。这个问题需要有一个满意的解决。

3.2 光

为什么这个问题如此难解决？这是由场景中光和对象之间交互的复杂性造成的。有关这一点的完全描述似乎超出了本书的范围。真正完整的研究可以阅读 Glassner（1989），本章部分内容就是基于这本书的。在这一小节中我们给出其中的一些问题。

可见光是波长大约在400nm~700 nm范围内的电磁辐射。波长引起对颜色的感知（在第4章中有更多关于这一点的阐述）。大家都知道光有波粒二象性。如果我们采用光的波模型，而且实验都是基于该原理进行的，那么所获得的结果与波理论是相容的。另一方面，如果我们采用的是光的粒子模型，那么实验结果也是与粒子理论相容的。

这种粒子称作光子，即一个在真空中以直线运动的能量包，它的速度为c（c为通常光速的符号，大约每秒 300 000 米）。每个光子携带的能量为 E，该值正比于它的频率：

$$E=hf \tag{3-1}$$

这里h叫做普朗克常数，f是光的频率。相应的波长反比于频率（频率越大波长越短）。事实 74 上：

$$\lambda f=c \tag{3-2}$$

波长(λ)乘以频率等于波的速度。

光子还有另外一个不寻常的性质，即它们彼此之间没有干涉现象发生——两条光线交叉彼此不相互影响。举例来说，到达我们眼睛的光子并不受到眼前其他交错光线的干扰。

光在一个空间范围内如何与表面交互是传导问题的一个实例。通常这与空间中移动粒子的分布有关——举例来说，道路系统中的车辆分布可以借助方程来研究，这种方程与光的传导方程是相似的（只不过运输问题增加了一些麻烦，因为包括车辆，所以还得考虑到粒子碰撞的效果）。

我们用Φ来表示**辐射能量**或在体积V中的**光通量**。光通量是单位时间内通过某个表面的能量（用瓦特作为测量单位）。能量正比于粒子流，因为每个光子都携带一定的能量。因此光通量可以看成是单位时间内的光子流量。

事实上能量是正比于波长的，所以要完全定义在一个体中的辐射能量，我们需要使用记号Φ_λ，即波长为λ的光波的辐射能量（更精确的表示用范围 $[\lambda, \lambda+d\lambda]$来表示）。我们暂时不提$\lambda$，用$\Phi$来表示一个特定的波长。从感知的角度看$\lambda$产生颜色的感知，而光通量引起亮度的感知。

现在考虑一个体中的总光通量。首先，这都是一种动态的平衡——虽然粒子在不断地流过某体，但是总的分布保持一个常数（场景的某个部分不会自发地变得忽明忽暗，其他东西也是一样）。对观察者来说，当场景中的一个光源被突然“打开”，光能量立即就分布到场景的各个角落，光照又保持稳定了。当然它不是真的瞬时发生，之所以看起来是这样完全是由于光速很快的缘故。

其次，能量守恒定律在起作用。对某个体来说，总的光能量输入一定等于流出的总能量和被该体内物质吸收的总能量之和。光可以以两种方式进入某个体：从外面流入（*入射*）或者是从该体内发射出（*发射*）。光从体内流出可能不受到来自该体内任何物质的影响，或者是受到该体内某些物质的影响经反射后流出（*出射*），也可能干脆就被这些物质所吸收（*吸收*）。

因此我们有方程：

$$\text{发射} + \text{入射} = \text{直接流量} + \text{出射} + \text{吸收} \tag{3-3}$$

我们要对此更进一步加以限制。设$\Phi(p, \omega)$是在点p处方向为ω的光通量，这里$p \in$
75 V、$\omega \in \Gamma$ 为一组感兴趣的方向集合（因此ω有形式(θ, ϕ)）。现在式（3-3）中每一项，除了发射项以外，都表示为概率形式。举例来说，吸收项表示为粒子在点p处沿着方向ω运动单位时间内被吸收的概率密度$a(p, \omega)$。因此在这点和这个方向总的被吸收密度为$a(p, \omega)\Phi(p, \omega)$。如果把所有点和所有方向的量加在一起，该体总的吸收量是：

$$\Phi_a = \int_\Gamma\int_V a(p,\omega)\Phi(p,\omega)\,\mathrm{d}p\,\mathrm{d}\omega \tag{3-4}$$

同样地，设$k(p, \omega, \omega')$是光子在点p以方向ω运动并偏转向方向ω'的概率。那么总的出射量是：

$$\Phi_o = \int_\Gamma\int_V\int_\Omega k(p,\omega,\omega')\Phi(p,\omega)\,\mathrm{d}\omega'\,\mathrm{d}p\,\mathrm{d}\omega \tag{3-5}$$

（记住Ω是单位球面上所有方向的集合。）

对入射量我们有一个相似的表达式Φ_i，但是ω和ω'要颠倒过来。

直接流量是整个表面S的光通量，该表面指的是V的整个边界表面，包括在任意方向上。这很容易写成积分形式：

$$\Phi_s = \int_\Gamma\int_S \Phi(p,\omega)\,\mathrm{d}p\mathrm{d}\omega \tag{3-6}$$

最后，发射密度函数定义为ε（p，ω），在点p处方向为ω发出的光通量密度（单位时间内的光子能量）。因此在体内的总发射量是：

$$\Phi_e = \int_\Gamma\int_V \varepsilon(p,\omega)\,\mathrm{d}p\mathrm{d}\omega \tag{3-7}$$

现在式（3-3）可以改写成如下形式：

$$\Phi_e + \Phi_i = \Phi_s + \Phi_o + \Phi_a \tag{3-8}$$

我们为什么要关心这些内容？它们与计算机图形学有什么关系呢？回答是令人惊讶的：如果我们知道了$\Phi(p,\omega)$，那么就能对计算机图形学中的光照问题给出完全的解决方案了。它告诉了我们在空间的某个体（V）内的各处，在我们所感兴趣的每个方向(Γ)上流动的光能。举例来说，假设在场景中有个虚拟眼睛或者相机，我们希望利用它来生成一幅图像（投影的虚幻世界），那么我们会求出进入透镜而且到达眼睛视网膜或相机胶片的所有光线（也就是一个集合$\omega \in \Omega$）。这个透镜对应一组点p。每个射到透镜表面的光线所具有的能量$\Phi(p,\omega)$决定了眼睛或照相机的响应。图像形成在虚拟的视网膜或“胶片”平面上，由此让人产生看虚拟场景的感觉。（当然，这如何实现将是本书其余部分的主题。） 76

我们如何能计算出$\Phi(p,\omega)$？我们可以试着去求解积分方程（式(3-8)）来得到$\Phi(p,\omega)$。这是很困难的，实际上无法达到。计算机图形学主要是通过一系列近似值来逼近式（3-8）的解。不同的需求对解的要求也不一样：一个满足实时性能的解和产生光照真实感的解是完全不同的，在这两种极端情形之间必须有个折衷。

3.3 简化假设

为了总能求出方程（式（3-8））的解，需要给出一些简化假设，这些简化假设通常在几乎所有的计算机图形学领域范围内使用。

波长独立性假设。我们通常假设具有不同波长的光波之间没有交互影响。因此式（3-8）可以针对不同的波长求解，然后估计在某一个点上光线波长分布，通过这些解的组合来获得具体的解。这要排除萤光。因为发射荧光过程中材料吸收一个波长的光，并在短时段内反射出另一个不同波长的光。

时间不变性假设。我们假设能量分布方程的任意解都不随时间而改变，除非场景本身发生了改变（例如某个对象移动到另外的位置）。这也排除存在磷光的情况，因为在这种情形中能量先被吸收，然后经过一个相对较长的时延之后再释放出来。

真空中光传导假设。一个非常重要的简化假设是光传导的空间是真空，除了场景中感兴趣的图形对象之外没有其他物质存在。换句话说，光运动的媒介是非参与性媒介，而不是参与性媒介。此时方程中的很多项都变得简单多了：吸收和出射只在表面的边界上发生。除了对象外没有其他物质能发射出光线。同样除了在对象的表面外没有散射或吸收现象发生。这意味着不受阻碍的光线（在两对象之间的光线——有时被称为“自由空间”）没有能量被吸收，没有沿该光线到处发散，沿着光线也没有物质粒子能自己产生额外的光能量。这对表现建筑物内部场景来说是一个合理的假设——但即使是在这种情况下灰尘的效果也不能被考虑在内。对于户外场景这是一个不合理的假设——它排除了雨、雾或任何由大气引起的光照效果。 77

对象均质性假设。当光子撞击一个表面时，它们的能量会有一部分被吸收，另外还有一部分被反射出去。对均质材料，如果我们考虑光的进入（或入射）方向和流出（反射）方向，它们之间的关系在对象的整个表面上处处都是一样的。这一点大大简化了 $k(p, \omega, \omega')$项的表示，因为对于这种对象来说，k是独立于表面 p点的。非均质材料没有这个性质。

3.4 光亮度

我们在上面的讨论中用到了光通量的概念，因为它是一种容易想像的概念——粒子穿行过一个体、粒子的数量和流速，以及粒子与体中的其他物质发生碰撞的效果等。然而，计算机图形学真正感兴趣的量不是光通量本身，而是由这个量导出的一个量，被称为光亮度。光亮度（L）定义为物体表面某个方向单位投影区域单位立体角的光通量。

设dA是表面区域，其法向为n，光从某一个与该法向夹角为θ的方向离开表面，经过的微分立体角为$d\omega$，如图3-1所示。如果离开“微分区域”dA的光亮度为 L，则对应的光通量为：

$$d^2\Phi = L\,dA\cos\theta\, d\omega \tag{3-9}$$

当我们让dA和$d\omega$都变得非常非常地小，$d^2\Phi$表示了在方向θ上的光通量。理解光亮度的另一个方法是将 $L(p, \omega)$看成一个被积函数，通过在立体角和投影区域上对它积分来获得该区域上的辐射能量（光通量）。

在图3-2中我们给出了两个小面片，dA和dB。假设r为它们之间的距离，从dA到dB的光亮度为 L。我们设Φ（dA，dB）为辐射能量的转移。那么有：

78

$$\Phi(dA, dB) = L\,dA\cos\theta_A\, d\omega_B \tag{3-10}$$

然而，使用式（2-23），我们有：

$$\begin{aligned} d\omega_B &= \frac{dB\cos\theta_B}{r^2} \\ \therefore \Phi(d(A, dB)) &= \frac{L\,dA\cos\theta_A\, dB\cos\theta_B}{r^2} \end{aligned} \tag{3-11}$$

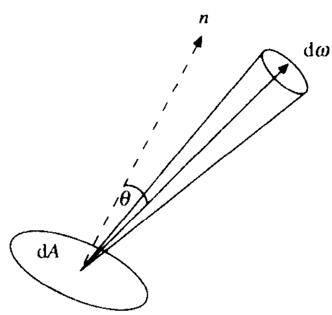


图3-1　光亮度是单位投影面积单位立体角上的光通量

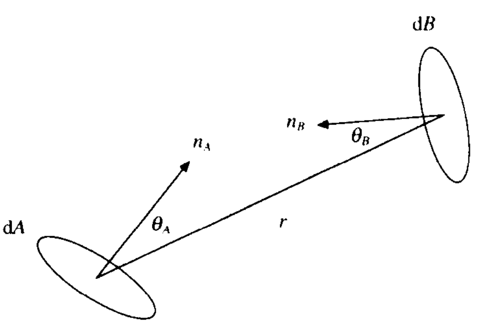


图3-2　两个无限小面片之间的光通量

这是光度测量的基本定律。通过重新整理并利用式（2-23），我们有：

$$\begin{aligned} \Phi(dA, dB) &= L\,dB\cos\theta_B\left(\frac{dA\cos\theta_A}{r^2}\right) \\ &= L\,dB\cos\theta_B\, d\omega_B \\ &= \Phi(dB, dA) \end{aligned} \tag{3-12}$$

式（3-12）说明光在相反方向上的光通量与原方向上光通量没有差别——根据等式的一般法则，当我们交换光的方向时，光能量不发生变化。

式（3-11）说明了光通量与距离的平方成反比——所以当两个区域间的距离变得更大时，光通量将逐渐减少。但光亮度不是这样：沿着某条光线的光亮度是个常数——它与距离光源的远近是没有关系的。

这引出了计算机图形学中的一条基本原理：我们不关心光子在空间体中的传导，而是从这个物理现象中抽象出光亮度的概念，即只考虑光所承载的能量。对于计算机图形学来说，基本的粒子不是光子和光子携带的能量，而是光线及其关联的光亮度。

对于计算机图形学来说，光能量有另外三个重要特性。第一个称为**光强度**。它是指单位立体角内的辐射能量（光通量）。因此，如果I是光强度，那么相关联的辐射能量由式（3-13）给出：

$$d\Phi=Id\omega \tag{3-13}$$

因此，与式（3-10）比较：

$$I=LdA\cos\theta \tag{3-14}$$ 79

这里 L 是对应的光亮度。

第二个重要特性是**辐射度**。它是从一个表面的单位面积上流出的光通量，通常用符号 B 来表示。如果B是与离开区域 dA 的能量相关的辐射度，那么光通量可以被重新计算如下：

$$d\Phi=BdA \tag{3-15}$$

辐照度是到达一个表面的单位面积上的光通量。它通常用E来表示，如果到达dA的辐照度为E，那么光通量为：

$$d\Phi=EdA \tag{3-16}$$

假设$L(p,\omega)$为沿着方向ω到达点 p的光亮度，那么根据式（3-9）我们有：

$$E(p,\omega)=\frac{d\Phi}{dA}=L(p,\omega)\cos\theta d\omega \tag{3-17}$$

3.5 反射

到目前为止我们讨论了场景中光能量的分布，并间接提及了光在表面的吸收和反射问题。这里我们概要地介绍一下在计算机图形学的光照模型中如何处理这些问题。假设一条光线射向表面的点p位置，其入射方向为ω_i。光能量将要反射的是个半球体，以在p点的切平面为底的一个半球。这一点的理由应该是清楚的：该半球包含了从p点可见的所有方向的集合，在这些方向上光线将不会受到来自表面的任何部分的遮挡，如图3-3。

下一个也许会问的问题将是有多少能量从反射方向ω_r离开表面？我们引进一个项$f(p,\omega_i,\omega_r)$，称**双向反射分布函数**（BRDF），通过它把在点p处方向为ω_r的反射光亮度和在点p处入射方向为ω_i的入射光亮度联系在一起。那么：

$$L(p,\omega_r)=f(p,\omega_i,\omega_r)E(p,\omega_i) \tag{3-18}$$ 80

对世界中真实表面的函数$f(p,\omega_i,\omega_r)$精确描述是一件极端复杂的任务，尤其是当表面为非均质的时候（这时f随点p而改变）。计算机图形学对真实世界的复杂性做进一步的抽象，主要使用两种理想化的材料属性，分别是**镜面反射器**和**漫反射器**，以及它们的混合。镜面反射

器是一个像镜子一样的表面，它将入射光线反射出去，入射光线的夹角与反射光线的夹角相等（如图3-4）。这些角度指的是在表面的 p点相对于表面法向的夹角。同时，相应于ω_i的矢量和相应于ω_r的矢量与表面法向位于同一个平面上。Richard Feynman在他的物理课讲义中给出了关于它的一个非常有趣的实验（Feynman et al.，1977）。漫反射器是一个“粗糙”的表面，它将入射光线的光亮度均匀地向以p点为中心的半球范围内的所有方向散布开去。此时我们可以证明:

$$f(p,\omega_i,\omega_r) \propto \frac{1}{\pi} \tag{3-19}$$

这里比例常数是材料的反射率（即入射光通量被反射出去的比例）。

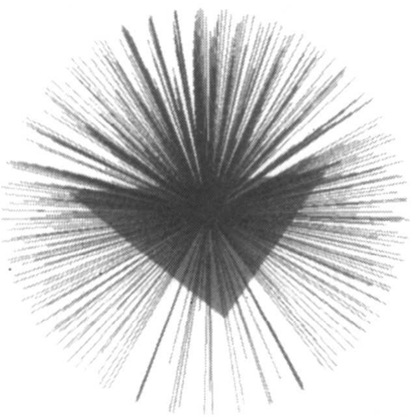


图3-3　光的发散范围是以平面上某点为球心的一个半球

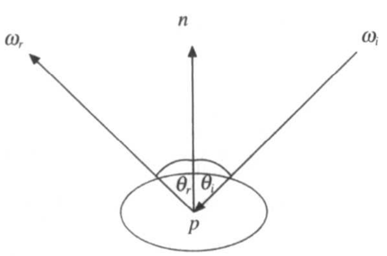


图3-4　镜面反射表面

我们还可以构造更复杂的 BRDF函数。比如平滑表面的反射，入射光线的反射范围是以理想镜面反射光线ω_r为中心的一个圆锥体。一个典型的BRDF是漫反射、镜面反射和平滑反射的组合。

其实我们隐含了一个假设，即假定表面是不透明的。然而，透明表面也一定要考虑在内。方式是非常相似的，只是光线要穿过不同的介质（比如从空气进入冰块内），在不同介质中的传播方向取决于介质的密度——更准确地讲是材料的折射系数。这将在第6章中做更详细的讨论。

3.6　光亮度方程

我们说过，如果可以求出在所有点和所有方向上的$\Phi(p,\ \omega)$，那么光照问题会完全得到解决，并且我们有关于这个函数的方程（式(3-8)）。其次我们给出了许多限制性假设，这是简化方程求解的重要条件。同时我们说明了该函数的目的不是在于$\Phi(p,\ \omega)$，而是$L(p,\ \omega)$。在前
81 面所有的简化假设（附加一些超出本书范围的其他假设）的基础上，使用先前引进的各项，我们能够从式（3-8）导出一个新方程，该方程使用光亮度而非光通量，更简单且更易理解。这就是*光亮度方程*，它给出了在表面上点p处沿着给定方向ω的光亮度。

这个方程说明了一个事实，即光亮度一定是两个量的总和。第一个量是直接从该点发射的光亮度的大小（如果有的话）。举例来说，该点本身可能就是表面上的一个光源。第二个量是从这个点反射的光亮度的大小。反射量的大小可以这样计算，用所有射入这个表面p点的光

线的总和（总辐照度）乘以 BRDF。因此，

$$光亮度 = 发射光亮度 + 总反射光亮度 \tag{3-20}$$

对任何入射方向ω_i其在方向ω上反射光亮度是辐照度乘以BRDF，利用式（3-17）和式（3-18），如下：

$$f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \mathrm{d}\omega_i \tag{3-21}$$

如果我们在p点为中心的包括所有入射方向的半球上做积分，则有：

$$总反射 = \int_{\Omega} f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \,\mathrm{d}\omega_i \tag{3-22}$$

因而对于出射光亮度 $L(p,\omega)$ 的光亮度方程是：

$$\begin{aligned} L(p,\omega) &= L_e(p,\omega)+\int_{\Omega} f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \,\mathrm{d}\omega_i \\ &= L_e(p,\omega)+\int_0^{2\pi}\int_0^{\pi/2} f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \sin\theta_i \,\mathrm{d}\theta\mathrm{d}\phi \end{aligned} \tag{3-23}$$ 82

这里 $L_e(p,\omega)$是发射光亮度。

这是一个关于光亮度函数L的积分方程，这本书的其余部分很大程度上都是关于如何求解该方程及求得 $L(p,\omega)$的内容。

我们先前限制p点要在表面上，但事实上这条限制是没有必要的。假设p不在表面上，那么因为在自由空间中光亮度沿着光线不发生改变，我们可以从p点开始以方向ω回溯光线，直到到达表面上点 p'，那么 $L(p,\omega)=L(p',\omega)$。

图3-5给出了光亮度方程的一个图示。考虑在点 p处任意一条入射光线。依照方程我们需要沿着这条光线计算光亮度。因此沿着入射方向ω_i的反方向回溯直到击中了另外一个表面上的p'点，求出光亮度$L(p',\omega_i)$。但是为了计算，我们需要再一次调用光亮度方程。换句话说，相应于每一个在图3-5中的入射光线（无穷多），都有一个相同的图，说明光亮度如何沿着那条光线产生的。

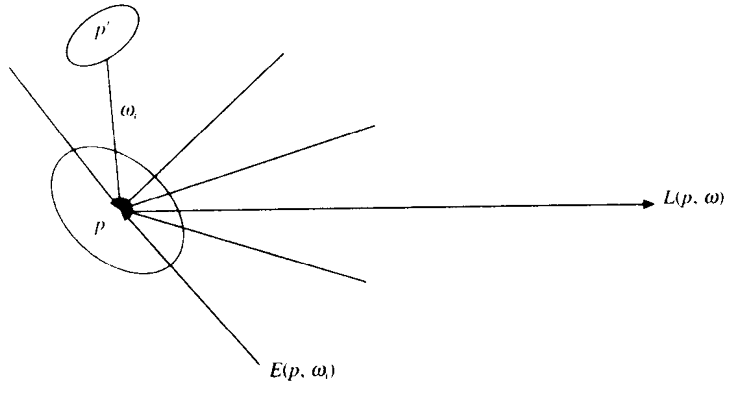


图3-5 光亮度方程的图解

通俗地说，光亮度方程强调的是光照的全局效果。你所看见的从书页上反射的光依赖于到书页上的入射光，以及书页表面的材料属性（即BRDF——该值反映了它是如何反射光的）。到书页的入射光依赖于你所位于位置的直接光源，也依赖于所有从你的环境中其他表面来的

间接光。而从那些表面上来的反射光同样又依赖于到达它们的直射光和从环境中的其他表面来的所有间接光，等等。

3.7　光亮度方程的解

在这一小节中我们考虑各种不同的求解光亮度方程的方法，实际上这个小节中的内容是本书中其余大部分内容的一个理论导引。

在计算机图形学应用中光亮度方程的重要性主要在于它暗含了所有可能的二维视图，即
83 表现场景的那些图像。这样问题就变成该如何抽取所需要的信息来构造这样的图像。从光亮度方程中抽取出二维投影图像的过程叫做**渲染**。

有两种方法：**视图独立解**和**视图依赖解**。视图依赖意味着光亮度方程的求解只求出形成图像所需要的光线集合。我们将在第5章及其后续章节中研究定义这种光线集合的技巧——暂时我们可以认为这个光线集合就是进入我们“眼睛”或照相机透镜的那组光线。这些是“眼睛”能惟一看得见的光线，是惟一引起视觉感知的光线。这种解之所以叫做“视图依赖”，这是因为如果观察条件发生改变，比如如果眼睛朝向另外一个方向，那么沿着眼睛看得见的光线计算光亮度的整个过程必须重新被执行。所以视图依赖解专门对一组(p，ω)计算$L(p，\omega)$，这里p位于代表透镜的表面上，ω对应于经过透镜光线的方向。

视图独立解专注于预计算$L(p，\omega)$的值，对场景中的所有表面和在尽可能多的方向上。当我们需要一幅特定的图像时，对应于经过透镜的那些表面和方向的(p，ω)被计算出来，其相应的$L(p，\omega)$就不再经过计算而是通过查表获得。因为求解$L(p，\omega)$无需考虑特别的视图，所以这种方法叫做“视图独立”。视图独立方法的优势在于产生场景的一个图像所需的时间是个常数，独立于场景的复杂度和特定视图。所需要的时间包括计算光线的方向和光线与透镜的相交时间，以及查对应的L值的时间。

对光亮度方程求解方法可以被独立地分为两种类型——依赖于解是否是局部的或全局的。**局部解**最多只考虑光源对对象的直接效果，不考虑对象间的反射。这一点彻底消除了式（3-23）中的递归操作。用来自光源的光线在入射方向上的求和计算有效地代替积分计算（这些光线的光亮度值是已知的，因而无需递归计算）。为了进一步简化，通常总是假设光源都是一些点，所以对于对象表面上的每个点，将会有惟一的一条光线代表来自一个特殊光源的入射光亮度。

全局解利用了光亮度方程的递归性质——换句话说，至少有某些类型的对象间相互反射要被考虑在内。所谓“光线跟踪”方法只考虑镜面表面的对象相互反射。所谓“辐射度”方法只考虑漫反射表面之间的对象相互反射。其他一些方法（通常称为“蒙特卡洛”方法）在一个很大的光线方向集合上用统计方法选取样本，并基于它们求得一个近似全局解。

表3-1是一些计算机图形学方法的分类，根据光亮度方程解的类型（局部解或全局解）和解的相位空间（视图依赖或视图独立）。我们依次对它们做简短的分析。

表3-1　光亮度方程的解类型

84

	局　部	全　局
视图依赖	“实时”图形	光线跟踪
视图独立	“平淡明暗处理”图形	蒙特卡洛路径跟踪
		辐射度
		蒙特卡洛光子跟踪

平淡明暗处理图形。光亮度方程的这类解是一种简单情形：它完全地忽略了积分项，所以方程变成了$L(P, \omega)=L_e(P, \omega)$。实际上这意味着每个对象有一个预先定义好的光亮度值（也就是它的颜色）。从任何视点看去，对象都只是在二维平面的投影，显示为它们自身的颜色。我们在第5章中使用这个模型。因为很清楚“解”的求得无需考虑任何视点信息，所以它是视图独立的。同时它又是“局部”的，因为显然解中没有递归，甚至没有来自主要光源的影响——除非我们将每个对象都看成是一个主要光源。

光线跟踪。对光亮度方程进行简化，只允许点光源存在，同时BRDF只考虑镜面反射。举例来说，假设在场景中只有一个（点）光源，那么每个$f(p, \omega_i, \omega)$只在ω和ω_i的惟一组合上是非零的——当入射角等于反射角的时候。因此在每个表面点处将有（至多）一条入射光线和一条出射光线。现在考虑“眼睛可见光线”，也就是一条以适当方向进入眼睛透镜的光线。在场景中找出表面上的一个点（p），该点为那条光线的出发点。沿着光线从p点到点光源，并由此计算$L_e(P, \omega)$（假设光的发射光亮度性质为已知，计算这个量是很容易的事）。这个量也包括一个称为“环境光”的成分和一个称为“漫反射光”的成分。所谓“环境光”是被假设用来表示来自间接光照的总背景光照，所谓“漫反射光”是表示表面的任意漫反射特性（它在光线跟踪中不是全局性的）。现在光亮度方程余下的部分是递归地进行光线跟踪，从p点开始沿着光线的反射方向跟踪光线直到碰到另外一个表面，然后同样沿着光线递归地计算光亮度。这个递归过程连续进行，直到光亮度的增量下降到某个预定的阈值以下。所以每条主光线（眼睛可见光线）衍生出一整棵描述递归跟随反射光的树，该树中每条光线将它们的光亮度带回到主光线。注意光线的路径是反向的——从眼睛向外到场景。因为方程的解只是针对进入眼睛的这组光线的集合，所以光线跟踪是视图依赖的。它又是全局的，因为解包括一个特殊的递归积分项这样一个特殊形式。有关光线跟踪的详细讨论将在第6章中展开，在第16章中还有另外一个讨论。

蒙特卡洛路径跟踪。这是一种与光线跟踪相似的方法——但它能生成光亮度方程的一个估计解，该解包括镜面反射和漫反射。因为主光线的跟踪是从眼睛向外到场景，它也遵循同样的原则。然而，与递归光线不同，递归光线所沿着的特殊路径是取决于镜面反射方向，这里所生成的光线是沿着一个随机选取的方向。这样假设与表面的相交点为p，BRDF随机地选 85
择一条光线计算。这样从一个相交点到另一个相交点直到增量可以忽略不计为止。对每条主光线整个过程被重复很多次，最后取结果的平均值。因此对于特别的视点所有解的空间进行了采样。这是一个视图依赖方法，因为它从眼睛可见的主光线开始，而且解只对一个特别的视图是有效的。很显然它是一个全局照明方法，因为对于整个光亮度方程它是一个随机解。这个方法将在第22章中有更详细的讨论。

实时图形。光亮度方程被进一步简化——递归成分被完全去掉。光源仍是点光源。只包括直接光照在内——这意味着积分由光源之和所取代，而且只计算到达表面上点p的局部贡献。这显然是一种局部解。这种方法不同于光线跟踪之处还表现在另外一个方面：在光线跟踪中，眼睛可见的主光线被跟踪到场景内；在实时图形中，所有对象都被“投影”到透镜表面——因而变成了二维实体（光线并没有明确使用）。在这样的二维投影空间中通过对对象边界上预计算的光亮度值插值来填充二维可见的对象实体区域。光线跟踪要对所有与光线相交的对象沿着光线路径搜索；实时图形通过直接投影对象到透镜表面来避免搜索操作——这样就没有了关于光线与对象相交的繁重计算。它是视图依赖的解，因为对象上的光照是和视点方向紧密相关的。关于实时图形的基本思想将在第9~13章中进一步阐述，在第23章中也有所涉及。

辐射度。这里对光亮度方程所采用的解法是让BRDF $f(p, \omega_i, \omega)$变成常数，不因为方向的改变而变化，并完全地除去方程中有关方向的那些项。从它的名字中可以看出，该方法也转换方程为另一种表达形式，用辐射度取代光亮度。现在让我们看一下它是如何消除所有方向上的考虑的，我们假设所有的表面都只是漫反射器。我们知道理想漫反射器对入射光线能量的散射在所有的方向上都是相等的——因此方向就无关紧要了。场景中的表面被分割为很
86 多小的表面单元。光亮度方程降为一系列线性方程（场景中每个小的表面单元对应一个方程），这里未知的是与表面单元相关的辐射度。因此这个方程的解是与每个小表面单元相关的辐射度，或者是每个小表面单元边界上点的辐射度。它是一个视图独立的解，因为只依据场景本身，而不与任何进入特定眼睛的光线集合相关。它显然是个全局解，因为确实考虑到了漫反射表面之间的相互关系。一旦辐射度计算完成，我们就可以利用实时图形的方法生成特殊视图。辐射度将在第15章中讨论。

蒙特卡洛光子跟踪。这种方法尝试求得光亮度方程的一般统计解，但与路径跟踪采用的方法相比是完全不同的。从光源开始到场景内部对大量随机分布的光线进行跟踪。每条光线跟踪到与它相交的最近一个对象，进一步产生的光线基于对象的 BRDF。本质上这是对光如何在环境中传导的直接模拟。然而，这是光传导的离散表示。当然每个对象与光线相交处是有限的，然而每个完整对象上的光亮度分布对于渲染是必须得到的。为了要克服这个困难，我们将对象分割成小的表面单元，这些小的表面单元可以用来估计对象表面的一个连续密度函数，以便估计在表面上从任意位置处和任意方向上的光亮度。这显然是个全局光照解。同辐射度一样，这里需要大量的预计算，用来估计环境中的光亮度分布。一旦这项计算完成，场景来自任意视点的渲染都能很快完成。有关这个方法的详细讨论将在第22章中给出。

3.8 可见性

光亮度方程有一个固有的问题，其计算量很大但是并不显而易见——这就是可见性问题。考虑照射到表面上点p的一条入射光线。这条光线来自哪里呢？显然是来自另外的一个表面。哪一个？光线跟踪必须明确地搜索光线与场景中所有对象之间的可能交点（有很多提高搜索速度的方法，将在第16章中讨论）。这里有一个可见性问题的例子：我们只对来自表面且对点 p 可见的入射光线感兴趣。（即使是这里也存在着一定的复杂性，因为经过透明对象的光线发生了弯曲，而且即使表面上p点没法直接看到，光线可能仍然对其有光亮度贡献。）可见性问题也在实时图形中存在，即使这里没有显式的光线跟踪。我们说过所有对象都投影到透镜的表面上。但是不是所有的对象都是眼睛所能看见的——而且从一个特别视点上看对象可能部分
87 或完全地被别的对象遮挡。很明显不能直接对对象进行投影——必须考虑特别视点对象之间的可见性关系。实时图形采用的方法将在第13章中讨论。有关可见性的一般问题将在第11章中详细讨论。

3.9 小结

本章涵盖了很多主题，主要包括：

- 说明了场景光照的高效计算被视为计算机图形学的核心问题——这是场景描述的几何形状的实质内容。
- 介绍了一些辐射测量的术语，包括光通量、辐射能量、光亮度、光强度、辐照度和辐射度。

- 说明了场景中光的分布如何用辐射能量场或光亮度描述。场是多维空间中点的函数。这里的空间是五维空间，每个点有形式(p, ω)。场景的相关信息都封装在函数$L(p, \omega)$中。
- 介绍了光亮度方程——只有满足该方程的$L(p, \omega)$才能满足场景对光亮度分布的需求。这是计算机图形学中的核心方程，在计算机图形学中所有的3D显示方法都是通过解这个方程来完成的——或（有时粗略地）使用该方程的简化形式。
- 渲染场景（二维投影）图像的过程就是从光亮度场中抽取相关信息的过程。这等同于求解光亮度方程的方法——我们已经简要地讲述了在计算机图形学中使用的几个不同方法。

本章篇幅较长。在学习完本书其余部分后应该再回头重读一遍本章的内容。此时如果你已经消化了这些内容，你将会对光照有一些理解，但是你仍然不知道该如何实际去做。在稍后时间里再读一遍本章，会对书中其余部分有更深刻的理解。

至此我们已经介绍了场景描述（或建模），以及光照的基本问题。人只是隐式的存在——需 88
要用他们的“眼睛”来渲染。在下一章中我们将人的视觉系统引入方程中。

第4章 颜色以及人对光的反应

4.1 引言：颜色作为光谱分布

前面一章我们使用了**辐射测量**领域的一些概念，例如光通量、光亮度等等——所谓辐射测量就是对光能量的测量。在这一章中我们将要研究的是*光度测定*——即与人的视觉系统反应相适应的测量方法。正如在第1章中所讨论的，虽然光能量进入我们的眼睛是视觉的一个必要条件，但这不决定我们能看见什么。在这一章中我们考虑人类视觉系统与颜色感知相关的方面。这一点对于深刻理解计算机图形学是至关重要的。举例来说，对场景计算出来的原始光亮度值渲染通常不仅仅是不可能的，而且不总是能给观察者所期待的图像效果。这一章中所采取的方法部分是基于Gomes 和 Velho（1997）的著作，还有一些材料来自 Glassner
89 （1995）的著作，其他的一些专门数据的来源可见本书参考文献。

从光源出发穿过某个表面的光子束通常是由不同能量的光子组成的。设λ代表波长，$n(\lambda)$是波长分布的密度函数，指每单位时间流过表面的波长位于λ为中心的小区间内的光子总量是$n(\lambda)\mathrm{d}\lambda$。从式（3-2）我们知道，光子所携带的能量反比于它的波长，因此对于以λ为中心的一个波长小区间内光子每单位时间经过表面的能量将是

$$\frac{Kn(\lambda)}{\lambda}\mathrm{d}\lambda \tag{4-1}$$

K为常数。

如果我们设

$$\Phi(\lambda)=\frac{Kn(\lambda)}{\lambda} \tag{4-2}$$

那么$\Phi(\lambda)$就是**光谱辐射能量分布**。

波长上的光谱分布是我们对颜色感知的**物理基础**。

设想我们用一个*分光光度计*采样一个特定光源（无论它是发射源还是个反射表面），该光源所发出的可见光波长范围大约在400nm~700nm之间，我们对每个波长测量光的辐射功率，得到如图4-1中所示的关于这个辐射测量量的光谱分布。用函数表示为：

$$\Phi(\lambda)\ (\lambda_a<\lambda<\lambda_b) \tag{4-3}$$

这里$\Phi(\lambda)\mathrm{d}\lambda$是每一个波长的能量（以瓦特为单位），$\lambda_a$和$\lambda_b$分别是可见光光谱的上限和下限。在可见光光谱上对这个函数积分我们就可以得到该光源的总辐射能量了（每单位时间）。因为所有的其他辐射测量量，例如辐射度和光亮度，主要都是由辐射能量导出的，所以我们
90 可以从这样的一个光谱分布中获得这些量。

当一个表面发出或反射某一个光谱分布的光时，它引起我们的感知，这种感知就是所谓的颜色。我们将用符号$C(\lambda)$表示一般的光谱分布，这里并不与所考虑的是哪一个辐射量有关。我们可以说我们所感知的颜色依赖于$C(\lambda)$，而$C(\lambda)$就是经过我们的视觉系统过滤的结果。不同类型的动物所拥有的视觉设备是不一样的，它们以完全不同的方式得到$C(\lambda)$。而且，两个

$C(\lambda)$经过视觉系统的处理会产生相同的彩色感觉，即使它们具有相当不同的分布。相同的视觉系统对截然不同的光谱分布会产生相同的感知效果，我们把这种现象称为条件等色。这在计算机图形学的颜色生成方面扮演着极为重要的角色。

只发某一个特别波长的光源称为单色。它所对应的$C(\lambda)$函数只在可见光谱内的一个单一点上不为零，所以该函数可以被认为是在这个点上的一条无限长的垂直直线，但“曲线之下区域”表示的能量仍然是有限的，如图4-2所示。数学上是用Dirac delta函数来表示的：

$$\delta(\lambda)=\begin{cases}\infty,\lambda=0\\0,\lambda\neq 0\end{cases} \tag{4-4}$$

$$\int_{-\infty}^{\infty}\delta(\lambda)\,\mathrm{d}\lambda=1$$

这里[-∞，∞]是实数轴，用R表示。这个函数还有性质如下：

$$\int_R f(t)\delta(x-t)\,\mathrm{d}t=f(x) \tag{4-5}$$

91

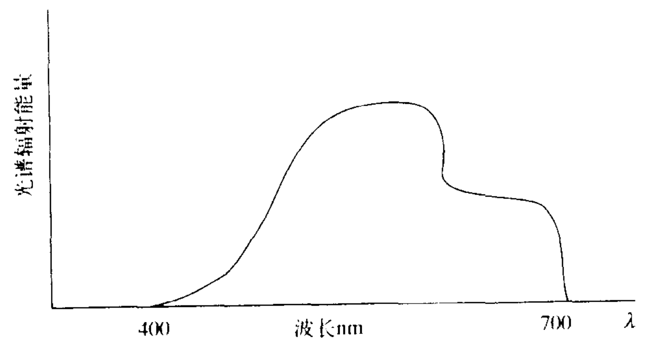


图4-1 光谱能量分布

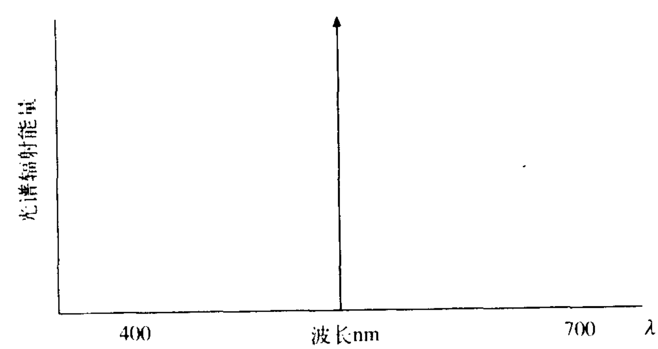


图4-2 单色光源的光谱分布

因此具有某个波长的纯单色光的光谱分布可以表示为：

$$C(\lambda)=\delta(\lambda-\lambda_0) \tag{4-6}$$

为了更好地理解单色光，我们回忆一下牛顿在1666年的发现。他发现我们平日所见到的白色太阳光是由整个可见光谱组成的。当太阳光经过棱镜时，或是更为壮观的景象——经过彩虹时，太阳光被分解开来，从中我们可以发现它是由整个可见光谱组成的——虽然特别突出的颜色有（以光波长从低到高依次）紫色、蓝色、青色、绿色、黄色、橙色和红色。

如彩图4-3所示。

与我们感知的白色光是由所有波长的光所组成的一样，彩色光也是这样一个混合体——只不过它并非是由光谱中所有波长的光均匀混合而成的，而是有其自身的一个特殊非均匀分布。

至此我们所讨论的主要是从某个光源所发出的能量。我们同样要对从表面反射出的能量给出同样的讨论。反射光遵从反射表面的 BRDF值，该值可以看成是依赖于波长的。因此不同波长的能量将会反射自不同的表面材质。由此可见能量反射自某个表面也当然会有一个光谱分布——即表面将会呈现出颜色。举例来说，如果光子来自白色光源，且根据表面的BRDF值，具有较高波长的光子比波长较低的光子在该表面得到反射的机会要大得多，则该表面将会显得较为发红。

在颜色的数学表示中，所有可见颜色空间等同于所有这些光谱分布集合$C(\lambda)$，这里$\lambda \in [\lambda_a, \lambda_b]$，$C(\lambda) \geqslant 0$，至少存在一个$\lambda$，满足$C(\lambda)>0$。任何能够解释（可视化）的真实物理系统或是能够产生彩色光的真实物理系统，都必须在物理世界的有限约束内以某种方式做到这一点。一个解释模型认为视觉系统是通过有限个通道过滤能量分布并使用这个能量分布来构造有限信息到大脑更高层处理单元——最后转换成视觉感知。感知不仅依赖于这个处理过程的物质方面，而且依赖于心理因素——例如期待、记忆、先验知识等等，作为这些感知加工的一部分。另一方面，一个物理的光发射器是通过对有限个不同强度的光源混合在一起的方法来产生彩色光的。这个有限光源的集合构成了发光系统的基础。从这一点我们能清楚地发现，人不能够"看"到所有可能的颜色。而且一个光发射系统也不能够发出所有可能的颜色。原因有两点，第一是对无穷颜色空间的离散采样，第二是试图以有限个基函数为基础来重建它。在下一小节中我们将讨论视觉系统、光发射系统，以及应用这些思想在计算机图形学中产生

92 颜色。

4.2 视觉系统简单模型

图4-4给出了人眼的一个示意图。光线经由瞳孔进入眼睛，孔径的尺寸是受虹膜控制的——对于黑暗环境它变得比较宽，对于明亮一些的环境则变得比较狭窄。光线的聚焦主要靠眼睛这个光学系统，它主要由虹膜、瞳孔、角膜所组成。角膜内充满晶状体，精细的调焦由透镜完成。透镜的厚度被睫状肌所控制。眼睛的后部有*视网膜*，它是由数以百万计的感光单元阵列所组成。在视网膜上视轴正对终点有一个小区域称为*黄斑中心凹*，在该黄斑区中感光单元特别密集，是视网膜上视觉最为敏锐的特殊区域。无论我们向什么地方看去，只有一处完全地聚焦在黄斑区上。之所以整个图像好像都是聚焦的，是因为我们的眼睛在不断地移动，使得不同的场景区域进入焦点。但是如果你一直盯着一个特别的场景看，同时注意一下该场景周围的区域的清晰程度，你会发现离中心区域越远的地方，图像越模糊。由视网膜层的处理结果被视神经传送到大脑皮质，最后由大脑皮层负责产生我们的视觉感知。

在视网膜上有两类感光单元——分别称为*杆状细胞*和*圆锥细胞*。大约有130 000 000个杆

93 状细胞、5 000 000~ 7 000 000 个圆锥细胞。杆状细胞分布在整个视网膜上，负责夜视（暗视），同时它也对运动高度敏感——尤其对外围的物体运动。圆锥细胞几乎完全集中在黄斑区及其附近；它们的任务是保证视觉灵敏度和彩色视觉，只在白昼工作（亮视）。

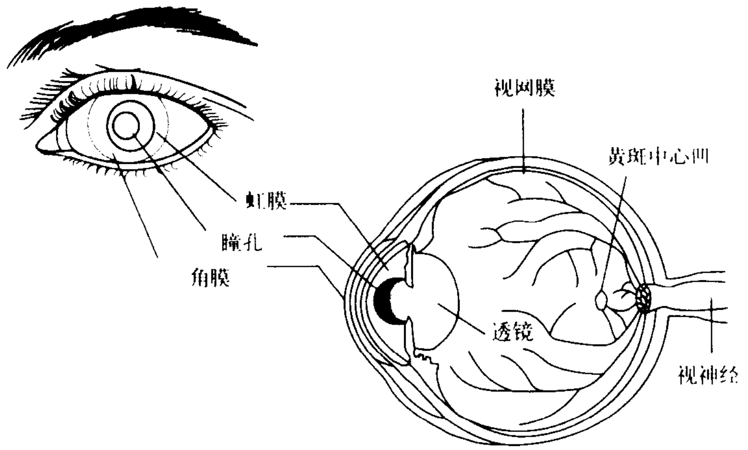


图4-4 人眼

根据它们对不同波长光的反应将圆锥细胞分为三种类型。第一类对长光波（红色）反应灵敏，称之为L型圆锥细胞；第二类对中等波长的光波（绿色）较为灵敏，称之为M型圆锥细胞；第三类对短波长光波（蓝色）比较灵敏，称之为S型圆锥体。它们对光波的反应函数如图4-5所示。图中2度表示观察者在实验中看彩色样本时的视角范围，通过这样的实验来获得这些曲线。

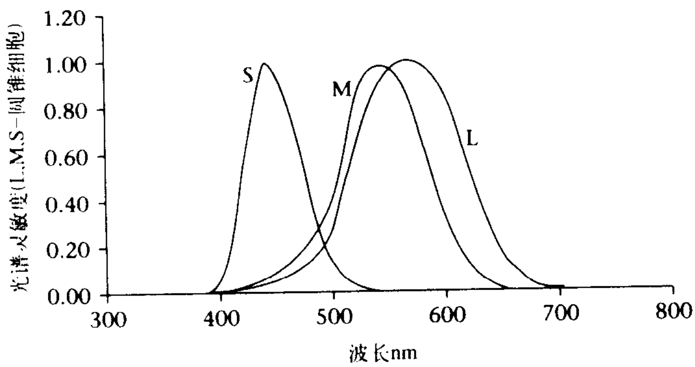


图4-5 2度圆锥细胞L、M和S标准化反应曲线（数据来自http://cvision.ucsd.edu/）

设三个反应函数分别是$L(\lambda)$、$M(\lambda)$和$S(\lambda)$，那么由光谱分布函数$C(\lambda)$表示的颜色经过这些反应函数的过滤，结果得到三个值：

$$
\begin{aligned}
l &= \int_{\Lambda} C(\lambda)L(\lambda)\,\mathrm{d}\lambda \\
m &= \int_{\Lambda} C(\lambda)M(\lambda)\,\mathrm{d}\lambda \\
s &= \int_{\Lambda} C(\lambda)S(\lambda)\,\mathrm{d}\lambda
\end{aligned}
\tag{4-7}
$$

这里，$\Lambda=[\lambda_a,\lambda_b]$。

这组LMS光谱反应函数将无穷维颜色空间映射到三维颜色空间，该三维颜色空间用三元组(l, m, s)的形式来表示。

从式（4-7）我们可以清楚地看出，完全不同的颜色仍然能被映射为相同的三元组——虽然光谱分布是截然不同的，但它们被感知为相同的颜色。如前面曾经提到过的，这被称作条件等色。如果我们将式（4-7）所表示的映射表示为 $LMS(C)=(l, m, s)$，那么当下式成立时 C_a
94 和 C_b 是条件等色：

$$LMS(C_a) = LMS(C_b)=(l, m, s) \tag{4-8}$$

4.3 发射器系统简单模型

发射器系统通过混合具有不同光谱分布光的能量束来生成彩色的光。物理上这等同于将多个光束混合在一起。每个光发出的能量的光谱分布是已知的，因此这样的光强度就可以任意改变。正如用三信道的视觉系统一样，我们同样假设发射器是由三种基本的光所构成，每种光的光谱能量分布为 $E_i(\lambda)$，i=1，2，3。那么发射器的光谱分布为：

$$C_E(\lambda)=\alpha_1 E_1(\lambda)+\alpha_2 E_2(\lambda)+\alpha_3 E_3(\lambda) \tag{4-9}$$

α_i是光强度，E_i构成基色（在数学上被称为基函数）。从式（4-9）我们能清楚地看到，由 $C_E(\lambda)$所产生的所有颜色构成的空间是所有可见颜色集合的一个真子集。换句话说——有一些可见颜色（光谱分布）不能够由这种物理系统产生。（注意，严格地说，一种颜色是观察某个光谱分布时所产生的感知，而并非光谱分布本身。）

对表示单色光基函数的选择具有特别的数学意义，同时在物理上具有很好的简便性。在1931年，国际照明协会（Commission Internationale de L'Eclairage[⊖]）定义了所谓的 CIE-RGB（红色、绿色、蓝色）基本光，如下各项所示：

$$\begin{aligned} E_R(\lambda) &= \delta(\lambda-\lambda_R), \lambda_R = 700\,\text{nm} \\ E_G(\lambda) &= \delta(\lambda-\lambda_G), \lambda_G = 546\,\text{nm} \\ E_B(\lambda) &= \delta(\lambda-\lambda_B), \lambda_B = 436\,\text{nm} \end{aligned} \tag{4-10}$$

它们称为CIE-RGB原色。三个光谱分布分别代表了纯红色、纯绿色和纯蓝色。这样，通过混合这些红色、绿色和蓝色原色，我们就可以生成所有可构造颜色集合中的任何一种了，这也就是所谓的 RGB 空间，它的定义如式（4-11）所示：

$$C_{RGB}(\lambda)=\alpha_R E_R(\lambda)+\alpha_G E_G(\lambda)+\alpha_B E_B(\lambda) \tag{4-11}$$

95 就物理意义而言，这种光发射器所发射的光是个混合光子束，每个光子的波长是纯红色、纯绿色和纯蓝色中的一种。这意味着不同的颜色可以用各种类型的光子强度产生出来。因此我们认为这种系统可以产生可感知颜色中足够大的有用子集。

4.4 产生可感知颜色

假设在真实场景中一个表面产生一种光谱分布为$C(\lambda)$的颜色。在计算机显示器上模拟该场景时，由$C(\lambda)$所表示的颜色也要在显示器上生成。由式（4-9）可知，一个实际的光发射器可能无法准确地生成一种任意光谱分布的颜色。然而，它可以用另一种光谱分布为$C_E(\lambda)$的颜色来代替那种颜色，这是由人类视觉系统的条件等色特性所允许的。因为人作为观察者可能无法区分由$C(\lambda)$和$C_E(\lambda)$表示的颜色，这种解决方法是够用的。

⊖ http://cvision.ucsd.edu/。

利用式（4-8）我们要求如下等式成立：

$$LMS(C) = LMS(C_E) = (l, m, s) \tag{4-12}$$

两种颜色在这种意义上构成条件等色，我们也可以将它写成：

$$C \approx C_E \tag{4-13}$$

注意在导出的一些数学表达式和基于感知的数学表达式中，可以将符号“≈”变成等号。举例来说，假设给定式（4-13），则一定有 $\int C(\lambda)L(\lambda)\,\mathrm{d}\lambda = \int C_E(\lambda)L(\lambda)\,\mathrm{d}\lambda$ ，否则它们不会成为条件等色。

从式（4-7）我们得到：

$$\begin{aligned} l &= \int_\Lambda C(\lambda)L(\lambda)\,\mathrm{d}\lambda = \int_\Lambda C_E(\lambda)L(\lambda)\,\mathrm{d}\lambda \\ m &= \int_\Lambda C(\lambda)M(\lambda)\,\mathrm{d}\lambda = \int_\Lambda C_E(\lambda)M(\lambda)\,\mathrm{d}\lambda \\ s &= \int_\Lambda C(\lambda)S(\lambda)\,\mathrm{d}\lambda = \int_\Lambda C_E(\lambda)S(\lambda)\,\mathrm{d}\lambda \end{aligned} \tag{4-14}$$

这里仅考虑L型圆锥细胞的方程，利用式（4-9），我们有：

$$\begin{aligned} \int_\Lambda C(\lambda)L(\lambda)\,\mathrm{d}\lambda &= \int_\Lambda (\alpha_1 E_1(\lambda) + \alpha_2 E_2(\lambda) + \alpha_3 E_3(\lambda))L(\lambda)\,\mathrm{d}\lambda \\ &= \alpha_1 \int_\Lambda E_1(\lambda)L(\lambda)\,\mathrm{d}\lambda + \alpha_2 \int_\Lambda E_2(\lambda)L(\lambda)\,\mathrm{d}\lambda + \alpha_3 \int_\Lambda E_3(\lambda)L(\lambda)\,\mathrm{d}\lambda \end{aligned} \tag{4-15}$$

这里$C(\lambda)$、$L(\lambda)$以及$E_i(\lambda)$是已知的。写成：

$$\begin{aligned} &\int_\Lambda C(\lambda)L(\lambda)\,\mathrm{d}\lambda = c_l \\ &\int_\Lambda E_i(\lambda)L(\lambda)\,\mathrm{d}\lambda = e_{il} \end{aligned} \tag{4-16}$$

96

把它们代入式（4-15），对式（4-14）中其他两个方程进行同样的应用，得：

$$\begin{aligned} \alpha_1 e_{1L} + \alpha_2 e_{2L} + \alpha_3 e_{3L} &= c_L \\ \alpha_1 e_{1M} + \alpha_2 e_{2M} + \alpha_3 e_{3M} &= c_M \\ \alpha_1 e_{1S} + \alpha_2 e_{2S} + e_{3S} &= c_S \end{aligned} \tag{4-17}$$

或

$$\begin{bmatrix} e_{1L} & e_{2L} & e_{3L} \\ e_{1M} & e_{2M} & e_{3M} \\ e_{1S} & e_{2S} & e_{3S} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} c_L \\ c_M \\ c_S \end{bmatrix} \tag{4-18}$$

这样我们得到三个线性方程，三个未知变量分别为α_1、α_2、α_3。因为原则上已知三个L、M和S反应函数和光发射器的基函数，所以我们能找到所需的光强度来产生所要的条件等色颜色。尤其是，如果指定了RGB原色，如在式（4-10）中的值，则方程的解给出了对于视觉系统来说与所要颜色相对应的“红色”、“绿色”和“蓝色”三色的强度。而且，此时我们能利用式（4-10）和式（4-5）中对RGB 原色的定义，有：

$$\int_\lambda E_R(\lambda)L(\lambda)\,\mathrm{d}\lambda = e_{1L} = L(\lambda_R)$$
$$\int_\lambda E_G(\lambda)L(\lambda)\,\mathrm{d}\lambda = e_{2L} = L(\lambda_G) \tag{4-19}$$
$$\int_\lambda E_B(\lambda)L(\lambda)\,\mathrm{d}\lambda = e_{3L} = L(\lambda_B)$$

对于中等波长和短波长反应曲线M和S有类似的结论。对图4-5做分析我们可以看出，在式（4-18）矩阵中的许多系数都接近于零，这将进一步简化计算。

总结：假设计算机图形学方法能够计算一表面（或者一条单独光线）的光谱分布，通过对不同波长的光束分别计算光亮度方程的解，从而构造光谱分布$C(\lambda)$的一个估计。同时假设对于某个特别显示器我们知道了RGB原色基函数，以及圆锥细胞的L、M和S反应曲线，那么我们可以解出式（4-18），以便得到显示原色应该具有的强度，从而能够得到$C(\lambda)$所表示的颜色。然而，事情远比这复杂。

4.5 CIE-RGB 颜色匹配函数

97 使用式（4-18）的问题在于它需要有关L、M和S型圆锥细胞反应函数的准确知识。这些不可能直接估计，只能从其他的信息中发现。另一种可用的方法是使用从相对直接的实验中估计出来的函数的方法。这些函数称为颜色匹配函数，对颜色理论具有至关重要的意义。

考虑一种波长为λ_0的单色，其光谱分布为$\delta(\lambda-\lambda_0)$。为了产生出该颜色，设光发射器所采用的基函数$E_i(\lambda)$的发光强度为$\gamma_i(\lambda_0)$。那么有：

$$\delta(\lambda-\lambda_0) \approx \sum_{i=1}^{3}\gamma_i(\lambda_0)E_i(\lambda) \tag{4-20}$$

假如我们以某种方式得到了大量波长中每种波长λ_0的$\gamma_i(\lambda_0)$值，而且知道了要产生该波长的一种纯色的条件等色，该如何配置原色的发光强度。注意这里把γ_i看作了波长的函数。我们将会很快做出对这些函数的物理解释，但是首先要说明它们的作用。

首先利用式（4-5）和式（4-7），我们发现视觉系统对这种波长λ_0的纯色的反应为：

$$\int_\lambda \delta(\lambda-\lambda_0)L(\lambda)\,\mathrm{d}\lambda = L(\lambda_0)$$
$$\int_\lambda \delta(\lambda-\lambda_0)M(\lambda)\,\mathrm{d}\lambda = M(\lambda_0) \tag{4-21}$$
$$\int_\lambda \delta(\lambda-\lambda_0)S(\lambda)\,\mathrm{d}\lambda = S(\lambda_0)$$

现在我们要将式（4-20）代入式（4-21）中，并利用式（4-16）（只给出第一行对于L的结果）：

$$\begin{aligned}&\int_\lambda\left(\sum_{i=1}^{3}\gamma_i(\lambda_0)E_i(\lambda)\right)L(\lambda)\,\mathrm{d}\lambda = L(\lambda_0)\\ &= \sum_{i=1}^{3}\gamma_i(\lambda_0)\int_\lambda E_i(\lambda)L(\lambda)\,\mathrm{d}\lambda = \sum_{i=1}^{3}\gamma_i(\lambda_0)e_{iL}\end{aligned} \tag{4-22}$$

因为λ_0是任意的，我们用λ替换它，乘以$C(\lambda)$并对λ积分得：

$$\sum_{i=1}^{3} e_{iL}\int_{\Lambda}\gamma_i(\lambda)C(\lambda)\,\mathrm{d}\lambda=\int_{\Lambda}C(\lambda)L(\lambda)\,\mathrm{d}\lambda \tag{4-23}$$

在式（4-23）的右侧利用式（4-15）和式（4-16）得：

$$\sum_{i=1}^{3} e_{iL}\int_{\Lambda}\gamma_i(\lambda)C(\lambda)\,\mathrm{d}\lambda=\sum_{i=1}^{3} e_{iL}\alpha_i \tag{4-24}$$ 98

合并同类项得：

$$\sum_{i=1}^{3} e_{iL}\left(\int_{\Lambda}\gamma_i(\lambda)C(\lambda)\,\mathrm{d}\lambda-\alpha_i\right)=0 \tag{4-25}$$

现对M和S反应函数进行同样的构造，得：

$$\begin{bmatrix} e_{1L} & e_{2L} & e_{3L} \\ e_{1M} & e_{2M} & e_{3M} \\ e_{1S} & e_{2S} & e_{3S} \end{bmatrix}\begin{bmatrix} \int_{\Lambda}\gamma_1(\lambda)C(\lambda)\,\mathrm{d}\lambda-\alpha_1 \\ \int_{\Lambda}\gamma_2(\lambda)C(\lambda)\,\mathrm{d}\lambda-\alpha_2 \\ \int_{\Lambda}\gamma_3(\lambda)C(\lambda)\,\mathrm{d}\lambda-\alpha_3 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4-26}$$

假定等式左面的矩阵满秩，因而是可逆的（在此不加以证明），我们有结论：

$$\alpha_i=\int_{\Lambda}\gamma_i(\lambda)C(\lambda)\,\mathrm{d}\lambda \tag{4-27}$$

换句话说，我们能通过对光谱分布与γ_i函数的积做积分来得到基色$E_i(\lambda)$的光强度水平(α_i)。因为它们给出了任意波长λ纯颜色相应的基颜色的光强度匹配，所以这些函数被称为颜色匹配函数。

颜色匹配函数可以通过简单实验来估计。先产生一束波长为λ_0的纯参考颜色的光。同时用三种基光谱的光束叠加来产生一束光，调整这些光束的强度直到所叠加生成的光正好与参考光束相匹配为止。记录下三种基光谱的强度值$\gamma_1(\lambda_0)$、$\gamma_2(\lambda_0)$、$\gamma_3(\lambda_0)$。现在对可见光谱范围内的各种波长值λ_1、λ_2、…、λ_n，重复相同的实验。这样就得到了对三个函数的估计。 99

图4-6 给出了CIE-RGB 颜色匹配函数。它们是用式（4-10）中的 RGB 三原色获得的，在390nm和830nm之间，采样间隔为5nm。因为观察者只能看见2度的视域，所以它们被称为“2度”颜色匹配函数。有10度匹配函数，但是在计算机图形学中一般使用2度匹配函数，因为当我们看屏幕的时候视域相对比较狭窄。

你会注意到有一些值是负的——这怎么可能呢？要知道这是彩色匹配，而且我们依赖条件等色来获得这些结果。现在假设我们正在试图匹配某一波长为γ_0的颜色。那么我们就要去求出$\gamma_1(\lambda_0)$、$\gamma_2(\lambda_0)$、$\gamma_3(\lambda_0)$，以便求得感知上的等色$\delta(\lambda-\lambda_0)$，即由三个光束生成的相同色（条件等色）。换句话说，我们需要：

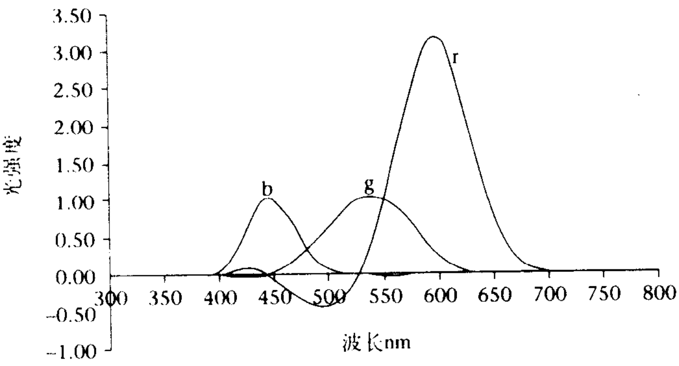
$$\delta(\lambda-\lambda_0)\approx\gamma_1(\lambda_0)E_R(\lambda)+\gamma_2(\lambda_0)E_G(\lambda)+\gamma_3(\lambda_0)E_B(\lambda) \tag{4-28}$$

也许有时这种匹配是不可能的。然而，假设把适当光强度的红色光波束加到参考波束上，然后试着去用剩余的两色做匹配。如果有一个匹配，那么有：

$$\delta(\lambda-\lambda_0)+\gamma_1(\lambda_0)E_R(\lambda)\approx\gamma_2(\lambda_0)E_G(\lambda)+\gamma_3(\lambda_0)E_B(\lambda) \tag{4-29}$$

因此原色中红色的系数变成负的了。同样地，如果不能通过叠加红色到参考光上，那么就用绿色去替代，然后用红色和蓝色去试着匹配。最后我们会用蓝色尝试，用红色和绿色波束来

匹配。通过这种方法，可见光谱中的每种纯色都可以由一个适当的红色、绿色和蓝色三原色获得匹配。



100 图4-6 2度RGB颜色匹配函数（Stiles and Burch，1955）

实际上所使用的值都经过与“白色点”进行比较——光束首先被调整，使得该物理系统给出“白”色。然后与一个特别的纯色相匹配的光强度被白色点的光强度除。如果构成白色点的三光束都达到了最大的光强度，那么匹配任何单色的光强度将是一个分数（值在0和1之间），表示每束光的光强度与最大光强度的比。

假如我们知道了来自表面$C(\lambda)$的颜色的光谱分布（比如我们通过求解一个虚拟场景中某表面的光亮度方程）。现在我们希望在显示器上产生一种颜色，保证给人的感觉好像与观察者在现实生活中看到的$C(\lambda)$一样。假设显示器采用CIE-RGB原色系统。那么我们使用式（4-27）计算光束光强度(α_i)，这里γ_i是RGB颜色匹配函数。我们将这些光强度传递到显示器上，这可以被当作一种行动，根据式（4-11）来产生$C(\lambda)$的条件等色。设CIE-RGB颜色匹配函数是：

$$\begin{aligned}\bar{r}(\lambda) &= \gamma_1(\lambda)\\ \bar{g}(\lambda) &= \gamma_2(\lambda)\\ \bar{b}(\lambda) &= \gamma_3(\lambda)\end{aligned} \tag{4-30}$$

那么：

$$\begin{aligned}C(\lambda) &\approx \alpha_R E_R(\lambda) + \alpha_G E_G(\lambda) + \alpha_B E_B(\lambda), \text{这里}\\ \alpha_R &= \int_\lambda \bar{r}(\lambda) C(\lambda)\,\mathrm{d}\lambda\\ \alpha_G &= \int_\lambda \bar{g}(\lambda) C(\lambda)\,\mathrm{d}\lambda\\ \alpha_B &= \int_\lambda \bar{b}(\lambda) C(\lambda)\,\mathrm{d}\lambda\end{aligned} \tag{4-31}$$

我们能通过数值积分求得RGB光强度值$(\alpha_R,\ \alpha_G,\ \alpha_B)$。

4.6 CIE-RGB色度空间

从前面一小节的讨论中我们能清楚看到，相应于每一个颜色$C(\lambda)$，在CIE-RGB空间中都存在一个条件等色$(\alpha_R,\ \alpha_G,\ \alpha_B)$。这种从光谱函数的无穷维空间到三维空间的映射显然是多对一的（否则就不会有条件等色存在）。这一小节中我们将研究这种三维颜色空间在下面的情形中“看起来”像什么。设想将每一个可见颜色转换到它相当的3D点上，而且该点依照对应

的（α_R、α_G、α_B）所表示的光强度着色。那么当我们连续移动穿越这个空间时，我们会不断地“看见”颜色在变化。换句话说，存在3D空间中的一个体对应于所有可能的可见颜色的集合，因而我们可以原则上创建3D空间中的体，该体被相应的颜色所着色，然后我们可以“穿行”其间（例如在虚拟现实中）。但是可视化这样的一个体是件非常困难的事情，而且可以看出是不必要的——因此可以放弃这一点，用二维面代替三维体。

首先考虑纯波长的单色。对任何这样一种波长为λ的颜色，我们有$C(\lambda)=\delta(\lambda-\lambda_0)$。将它代入式（4-31）中得： 101

$$(\alpha_R(\lambda_0),\alpha_G(\lambda_0),\alpha_B(\lambda_0))=(\bar{r}(\lambda_0),\bar{g}(\lambda_0),\bar{b}(\lambda_0)) \tag{4-32}$$

对任何$\lambda_0\in[\lambda_a,\lambda_b]$。

因为λ_0可以是所有可见波长，式（4-32）形成3D空间中的一条曲线，这条曲线表示所有纯色的条件等色。实际上我们不是在3D空间中画这样一条曲线，代之以通过原点将该曲线投影到一个平面上，并绘制这个投影曲线。经常取曲线的投影平面为如下平面：

$$\alpha_R+\alpha_G+\alpha_B=1 \tag{4-33}$$

这里α是x、y、z的三维空间坐标。

假设(α_R、α_G、α_B)是该曲线上的一个点。那么从原点到这个点的直线参数化方程是：

$$(t\alpha_R, t\alpha_G, t\alpha_B)，对所有的t>0 \tag{4-34}$$

这里该直线与平面（式（4-33））相交，一定有：

$$t\alpha_R+t\alpha_G+t\alpha_B=1 \tag{4-35}$$

因此有：

$$t=\frac{1}{\alpha_R+\alpha_G+\alpha_B} \tag{4-36}$$

因此任何点（α_R（λ_0）、α_G（λ_0）、α_B（λ_0））在平面上的投影将是：

$$\left(\frac{\alpha_R(\lambda_0)}{\alpha_R(\lambda_0)+\alpha_G(\lambda_0)+\alpha_B(\lambda_0)},\frac{\alpha_G(\lambda_0)}{\alpha_R(\lambda_0)+\alpha_G(\lambda_0)+\alpha_B(\lambda_0)},\frac{\alpha_B(\lambda_0)}{\alpha_R(\lambda_0)+\alpha_G(\lambda_0)+\alpha_B(\lambda_0)}\right)$$

到二维的正射投影忽略了表达式中第三个坐标（蓝色），结果如图4-7所示。这就得到了所谓的CIE-RGB“色度空间”。曲线上的每个点是基于 RGB 光强度的组合，用来生成所要的具有特定波长的纯色。三个这样的波长在曲线上标出。 102

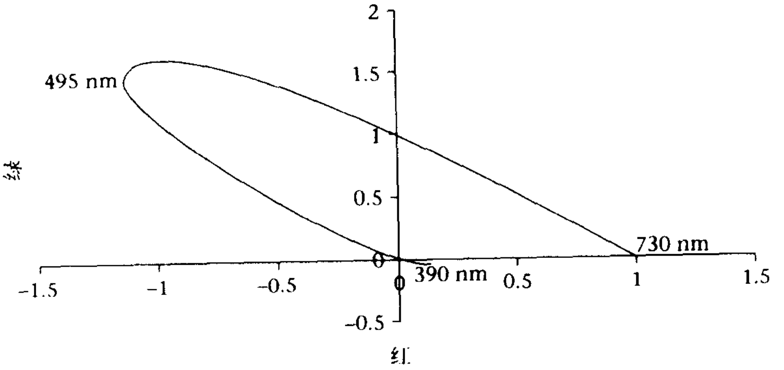


图4-7 CIE-RGB色度空间（白色点未标注）

让我们进一步研究一下这个空间。现在假设$C_1(\lambda)$和$C_2(\lambda)$是两个不同可见颜色的光谱分布，同时设在 RGB 颜色空间中对应的点是：

$$\alpha_1 = (\alpha_{R1}, \alpha_{G1}, \alpha_{B1})$$
$$\alpha_2 = (\alpha_{R2}, \alpha_{G2}, \alpha_{B2}) \tag{4-37}$$

那么

$$\alpha_1 = \left(\int_\Lambda \bar{r}(\lambda)C_1(\lambda)\,d\lambda, \int_\Lambda \bar{g}(\lambda)C_1(\lambda)\,d\lambda, \int_\Lambda \bar{b}(\lambda)C_1(\lambda)\,d\lambda \right)$$
$$\alpha_2 = \left(\int_\Lambda \bar{r}(\lambda)C_2(\lambda)\,d\lambda, \int_\Lambda \bar{g}(\lambda)C_2(\lambda)\,d\lambda, \int_\Lambda \bar{b}(\lambda)C_2(\lambda)\,d\lambda \right) \tag{4-38}$$

在连接α_1和α_2的直线上的所有点都可以表示为参数化形式：

$$(1-t)\alpha_1 + t\alpha_2, \text{对所有} t \in [0, 1] \tag{4-39}$$

从式（4-38）我们可以看到，这是$(1-t)C_1(\lambda)+tC_2(\lambda)$的条件等色。假若$t$在0和1之间（闭区间），上式代表了一个可见颜色（如果$t<0$ 或$t>1$，那么光谱分布就变成负值了，不能够表示一个可见颜色）。当我们将式（4-39）所代表的直线段投影到平面$\alpha_R+\alpha_G+\alpha_B=1$上时，该投影是一条线段，线段的两个端点分别为$\alpha_1$、$\alpha_2$两点的投影。它的含义是所有的可见颜色都投影在这个平面上，它们的投影都在曲线内（或在边界上），如图4-7中所示。换句话说，在该曲线边界上的任意两点的连线上的任意一点都代表了一个可见颜色。因为在曲线内的任一点都在无数条这样的可能直线上，曲线内部的所有点都对应于可见颜色。因此所有可见颜色的集合映射到曲线的内部和边界。

我们现在讨论投影平面的感知解释。回忆一下视觉系统的L、M和S圆锥细胞反应函数，这些定义了三种光感圆锥细胞单元对可见光谱范围内的不同波长光的反应。视觉系统作为一个整体，它总的反应可以考虑为对这三个反应曲线的加权平均：

$$V(\lambda) = \beta_1 L(\lambda) + \beta_2 M(\lambda) + \beta_3 S(\lambda) \tag{4-40}$$

β_i的特殊值构成视觉系统的一条光谱发光效率曲线，由$V(\lambda)$表示，这是按照CIE标准定义的。该函数的曲线图如图4-8所示。

103

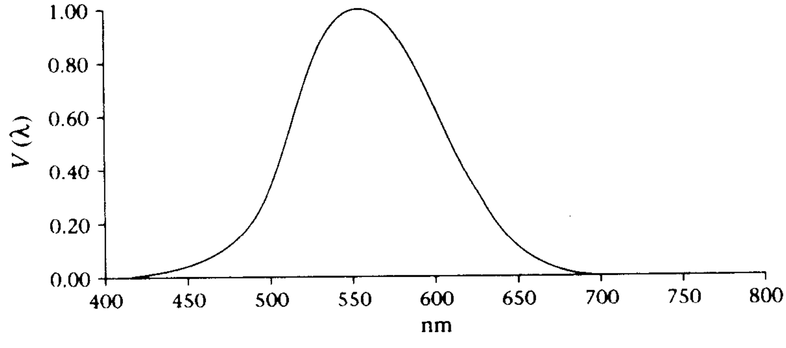


图4-8 光谱发光效率函数

现在给定任意光谱分布$C(\lambda)$，视觉系统对该颜色的总反应可以使用发光效率函数通过调制光谱分布计算出来：

$$L(C)=K\int_{\lambda}C(\lambda)V(\lambda)\,\mathrm{d}\lambda \tag{4-41}$$

这里K是一个给定的常数，以便获得相关量的正确度量单位。（以下我们一般不写K）。尤其是，如果$C(\lambda)$是一个光亮度光谱分布，K=680流明/瓦特，亮度用"每平方米坎德拉"计量，那么L被称作反应视感度。我们先前曾经讨论过视觉系统对光亮度的反应，事实上这种反应是通过调制发光效率函数得到的光亮度。当我们应用该函数于任何类型的光谱能量分布的时候（例如光通量分布而非光亮度分布），该函数将辐射计量量转换为等价的光度测量量。前者是物理能量测量量，而后者是与视觉系统的反应相关的量。

回到式（4-11），在那里我们看到了光谱分布如何能从原基色构造出来，并求出相应的亮度：

$$\begin{aligned}&C(\lambda)\approx\alpha_R E_R(\lambda)+\alpha_G E_G(\lambda)+\alpha_B E_B(\lambda)\\&\therefore L(C)=\alpha_R\int_{\lambda}E_R(\lambda)V(\lambda)\,\mathrm{d}\lambda+\alpha_G\int_{\lambda}E_G(\lambda)V(\lambda)\,\mathrm{d}\lambda+\alpha_B\int_{\lambda}E_B(\lambda)V(\lambda)\,\mathrm{d}\lambda\end{aligned} \tag{4-42}$$

这又可以写成如下形式：

$$L(C)=\alpha_R l_R+\alpha_G l_G+\alpha_B l_B$$ 104

现在l_R、l_G和l_B是常数，它们完全取决于主基色和发光效率函数。满足式（4-42）的所有可能值（α_R、α_G、α_B）的集合构成三维空间中的一个颜色平面。但是对于所有可见光，在这个平面上的每个点都具有相同的亮度$L(C)$，然而只有一点与颜色$C(\lambda)$相对应。那么在如此一个具有固定亮度的平面上，什么是改变的呢？它一定是我们平常所感知的有关颜色本身的一些属性。

为了避免用词混乱，我们称颜色的这个属性为*色度*。因此颜色包含了两种截然不同而又独立的属性：一个是标量，我们称之为亮度；另一个是两维矢量，我们称之为色度。

让我们重写式（4-42），对它稍加改变：

$$\alpha_R l_R+\alpha_G l_G+\alpha_B l_B=L \tag{4-43}$$

记住l_R、l_G和l_B都是常数。（α_R、α_G、α_B）是变量，代表坐标轴。如果我们对方程中各项都乘以一个常数t，$t>0$，当然方程仍然满足，只是所表示的平面的亮度值变成了tL。设$(\alpha_R',\alpha_G',\alpha_B')$是颜色空间满足式（4-43）的一个特殊点。因为$t$在改变，$(t\alpha_R',t\alpha_G',t\alpha_B')$的轨迹是一条从原点出发经过点$(\alpha_R',\alpha_G',\alpha_B')$的直线。亮度沿着这条直线而改变（随着$t$的变大而逐渐增加），但是色度在这条直线上是不变的，保持为常数。因此该直线与平面$\alpha_R+\alpha_B+\alpha_B=1$求交，给出了一个用来表示色度性质的二维坐标系统，这就导出了在上面讨论过的色度图。

4.7 CIE-XYZ色度空间

图4-7中的CIE-RGB色度图是相当奇怪的。从中可以看出可见颜色的很大一部分实际上是不能通过光发射器的CIE-RGB原色来生成的。在该图的第一象限（或相应的整个3D颜色空间的第一象限）外面的颜色都不能用基色来实现——因为它们对应于负的光强度。

由此引出了这样的一个问题，即是否有一组替代原色能使我们得到更满意的结果——特别是能让所有可见颜色都包含在图的第一象限里面，而且对应的匹配函数处处非负。所出现的特殊色度图依赖于对主基函数的选择。首先，我们考虑在由不同基函数所构成的系统之间的关系，然后介绍一种CIE-XYZ系统，它具有所希望的性质，即让所有色度都位于第一象限内。

假如F和E是两组不同的主基函数，所以对于一给定的颜色$C(\lambda)$，我们利用式（4-9），有：

$$C(\lambda) \approx \alpha_1 E_1(\lambda) + \alpha_2 E_2(\lambda) + \alpha_3 E_3(\lambda)$$
105
$$C(\lambda) \approx \beta_1 F_1(\lambda) + \beta_2 F_2(\lambda) + \beta_3 F_3(\lambda) \tag{4-44}$$

现在应该可以将一个系统中的基函数用另外一个系统的基函数来表示。举例来说：

$$F_1(\lambda) = \sum_{j=1}^{3} \alpha_{1j} E_j(\lambda)$$
$$F_2(\lambda) = \sum_{j=1}^{3} \alpha_{2j} E_j(\lambda) \tag{4-45}$$
$$F_3(\lambda) = \sum_{j=1}^{3} \alpha_{3j} E_j(\lambda)$$

将上式写成矩阵形式：

$$\begin{bmatrix} F_1(\lambda) \\ F_2(\lambda) \\ F_3(\lambda) \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} E_1(\lambda) \\ E_2(\lambda) \\ E_3(\lambda) \end{bmatrix} \tag{4-46}$$

或

$$F(\lambda) = AE(\lambda) \tag{4-46'}$$

现在

$$\alpha E(\lambda) = \beta F(\lambda)\text{，这里有 } \alpha = (\alpha_1, \alpha_2, \alpha_3), \beta = (\beta_1, \beta_2, \beta_3)$$
$$\therefore \alpha E(\lambda) = \beta A E(\lambda)\text{，从而有：} \tag{4-47}$$
$$\alpha = \beta A$$

又根据式（4-47）：

$$\alpha_j = \sum_{i=1}^{3} \beta_i \alpha_{ij}$$
$$\therefore \int \gamma_{E_j}(\lambda) C(\lambda) \mathrm{d}\lambda = \sum_{i=1}^{3} \int \gamma_{F_i}(\lambda) C(\lambda) \mathrm{d}\lambda \, \alpha_{ij} \tag{4-48}$$
$$= \int \sum_{i=1}^{3} \gamma_{F_i}(\lambda) \alpha_{ij} C(\lambda) \mathrm{d}\lambda$$

这里$\gamma_{E_j}(\lambda)$和$\gamma_{F_j}(\lambda)$是两个基色的颜色匹配函数。

从而有：

$$\gamma_{E_j}(\lambda) = \sum_{i=1}^{3} \gamma_{F_i}(\lambda) \alpha_{ij} \tag{4-49}$$

从式（4-46）、式（4-47）以及式（4-49）中可以看到，如果我们知道一个基色的匹配函数，并且知道从一个基色到另一个基色的变换矩阵，那么就能够轻松地求得第二个基色的匹
106 配函数。而且，这些变换都是线性变换。

CIE定义了一种被称为XYZ颜色系统的原色。这个名字源于三个基函数，分别是$X(\lambda)$、$Y(\lambda)$和$Z(\lambda)$。这种定义使得X和Z具有零亮度，而Y的颜色匹配函数等于发光效率曲线V。由此可见，XYZ系统的原色不是真正的颜色（为了让X和Z具有零亮度，它们在某些范围内就必须

为负值)。

在式(4-45)中设E是RGB系统，F为新的XYZ系统。那么可以证明变换可以用下式定义：

$$A=\begin{bmatrix}0.489989 & 0.310008 & 0.2\\ 0.176962 & 0.81240 & 0.010\\ 0.0 & 0.01 & 0.99\end{bmatrix} \tag{4-50}$$

换句话说，给定一个CIE-RGB颜色，其等价的CIE-XYZ描述能通过在式(4-45)中那样前乘矩阵A得到。其逆矩阵将CIE-XYZ颜色转换为等价的CIE-RGB颜色：

$$A^{-1}=\begin{bmatrix}2.3647 & -0.89658 & -0.468083\\ -0.515155 & 1.416409 & 0.088746\\ 0.005203 & -0.014407 & 1.0092\end{bmatrix} \tag{4-51}$$

函数$X(\lambda)$、$Y(\lambda)$、$Z(\lambda)$应用先前的方式定义了一个颜色空间(事实上，它所定义的空间是完全相同的空间，只不过是用不同的坐标系统来表示的)。因此任何光谱分布$C(\lambda)$可以表示成如下形式(事实上更准确的说法是条件等色)：

$$C(\lambda)\approx X\cdot X(\lambda)+Y\cdot Y(\lambda)+Z\cdot Z(\lambda) \tag{4-52}$$

X、Y和Z皆为常数。

所得到的色度图如彩图4-9所示。

颜色匹配函数$\bar{x}(\lambda)$、$\bar{y}(\lambda)$、$\bar{z}(\lambda)$在图4-10中给出。注意，它们是处处非负的，而且函数$\bar{y}(\lambda)$同光谱发光效率曲线V相一致。这些函数已经被制成了表格[⊖]。因此，给定一个光谱分布函数$C(\lambda)$，我们从式(4-53)求出系数(X, Y, Z)： 107

$$\begin{aligned}X&=\int\bar{x}(\lambda)C(\lambda)\,\mathrm{d}\lambda\\ Y&=\int\bar{y}(\lambda)C(\lambda)\,\mathrm{d}\lambda\\ Z&=\int\bar{z}(\lambda)C(\lambda)\,\mathrm{d}\lambda\end{aligned} \tag{4-53}$$

注意Y值是发光亮度。对应的色度值从下式求得：

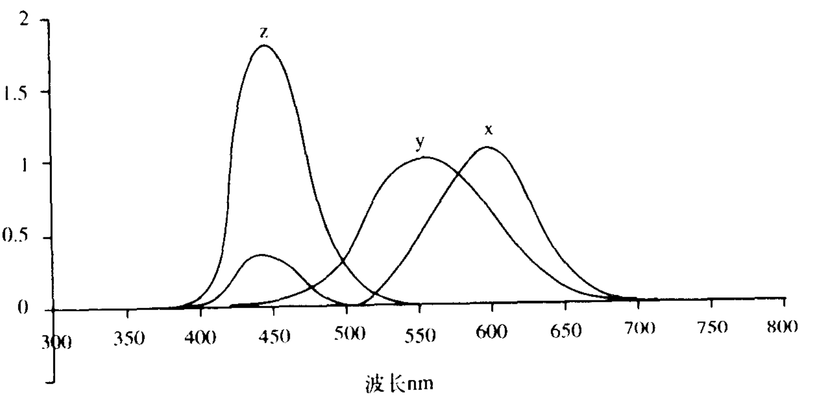


图4-10 2度XYZ颜色匹配函数

⊖ 参见http://cvision.ucsd.edu/。

$$x=\frac{X}{X+Y+Z}$$
$$y=\frac{Y}{X+Y+Z} \qquad (4\text{-}54)$$
$$z=1-x-y$$

4.8　CRT显示器的一些特性

CIE-XYZ颜色空间是标准且与设备无关的颜色空间，这种构造是为了要让所有可见颜色都能通过三原色的叠加来生成。然而，XYZ 空间的原色是虚构的，所以这个空间无法由光发射器系统对有限原色组合物理地实现。然而由于XYZ空间是设备独立的，而且能表现所有的可见颜色（相当于人的视觉系统的条件等色），所以它很适合于计算机图形学。既然 XYZ 颜色不能够被显示，那么就存在这样的一个问题：该如何将它转换成可以在真实显示器上显示的颜色。

在分析这个问题之前，在这一小节中请让我们简要地描述一下典型的阴极射线管（CRT）的特性，这是当前主要采用的显示器系统。有关它的全面介绍请参考 Glassner（1995）。CRT 显示器可能被看成二维的元素阵列，每个元素称为**像素**，英文写做pixel（由“picture elements”缩写而成）。像素是显示器的最小单元，每个像素的颜色可以独立地设置——任何像素颜色的确定都不受其他像素的影响。显示屏的内部涂有一层能发光的荧光粉。对于每个像素都有三个荧光点，它们排列成某种结构。当有一个电子束撞击到荧光点上时，它就会发出光来。三个荧光点中的一个能发出“红色”光，其他两个分别发出“绿色”光和“蓝色”
108 光。每个像素的三个荧光点彼此靠得很近，它们所发出的光组合在一起形成一个混合颜色。CRT 有三个电子枪，它们的位置是精心安排的，以便于来自每只枪的光束只能打在每个像素区域里具有特定颜色的那个荧光点上。电子束的电压决定了从三个荧光点发出的光的强度，这样就能得到不同的RGB 颜色组合。

荧光点具有持久性。电子束撞击到荧光点引起它释放能量，荧光点从开始释放能量直到能量释放完需要一段时间。一般每个荧光点至少每秒钟内要刷新60次（60赫兹）。这是通过电子束一遍一遍地在水平方向和垂直方向扫射荧光点达到的，水平的扫射从左至右逐个撞击像素行，垂直的扫射从顶部开始向下逐行逐行地进行，完成一遍之后从顶部开始重新执行。这个刷新过程的执行模式通常叫做**光栅扫描**——当像素的一个水平阵列刷新完成，电子束移到下一行的开始处，开始下一行的刷新，当最底端的一行的最后一个像素被扫描完时电子束回到左上角并开始新一轮的扫描。电子束返回到下一行开始处这段时间叫做“水平回扫时间”，电子束从屏幕的右下角返回到左上角的这段时间叫做“垂直回扫时间”。如果完成一次全屏幕刷新的时间长于荧光能量释放的持续时间，那么图像将会发生闪烁。刷新频率至少要达到 24 赫兹才能保证人的视觉系统不会觉察到刷新的间隔，保证看到一幅连续的图像。实际上，今天的显示系统所具有的刷新频率远远高于这个数字。

显示器的像素数目通常称为**分辨率**。显然分辨率越高，刷新周期就要越短，以便避免闪烁（一般目前的显示分辨率为1280 × 1024，第一个量表示的是水平像素数目，第二个量表示的是垂直像素数目）。像素是通过它在显示屏幕上的坐标定位的。通常原点（0，0）位于左上角，坐标（x，y）表示自左侧的第x列上且自上方第y行上的像素。注意这里的显示空间是与标准的数学表示是“颠倒”的，标准数学坐标表示中Y坐标是自下而上的。然而，设垂直像素

数目是 N，如果 (x, y') 是相应于 (x, y) 的像素在标准数学表示中的坐标表示，那么有：

$$(x, y') = (x, N-1-y) \tag{4-55}$$

在整本书中我们都假设坐标系遵从标准的数学表示，也就是让Y坐标从底部向上（如果你没有记住这一点，没有在程序中做相应的转换，你初次得到的图像可能就是颠倒的）。

像素有时被看成是一个无穷小的点。然而，从上述的介绍中我们可以清楚地发现像素是有确切大小的，它所发出的光会影响到周围的像素。将像素看成“点”在图像显示上会引起严重的问题——称为*走样*。举例来说，除了在水平方向、垂直方向和45度方向的情形外，在两两像素之间都不存在“真正的直线”。只能选取最接近直线的那个像素，而且要调整它们的发光强度，让人感觉不出来量子化所造成的锯齿形状。通常一个像素的显示是对某个连续图像的采样，同时理想的过滤技术要被用来保证获得对真实图像的一个最好逼似。 109

是什么信息用来确定每个像素上电子束的电压值？CRT 显示器当然是受CPU的控制。内存中有一个与此密切相关的部分，被称为*帧缓冲区*，该内存区域可以被看成是二维的存储单元阵列。我们暂时假设在帧缓冲区中的每个存储单元和每个像素之间存在一对一的映射。现在帧缓冲区中每个单元的大小就对显示器的颜色分辨率起到了决定性的作用。帧缓冲单元必须分别对红色、绿色和蓝色保留多个位。比方说对于红色，它所对应那几个位的值确定了像素上电子束撞击其“红色”荧光点时的电压。位数量越大，它可以表示的颜色范围也就越大。这里包含一个量子化过程。通常红色、绿色和蓝色三原色每个都对应有8个位，所以每个电子束的电压范围只能有 256 个不同值。因此我们暂时认为帧缓冲区中对应于一个像素的区域由24个位组成，分为三个字节。每个字节确定了相应电子束撞击像素时的电压值，因此有$2^8 \times 2^8 \times 2^8$=16 777 216种颜色可以被这种显示设备显示出来。有时将这种系统称为*真彩色*。

实际上，帧缓冲区阵列可能远大于实际显示器上像素的数目，这意味着显示器在任何时候都可以被看成是帧缓冲区上的一个“窗口”。同时，由于像素中还可能储存着其他一些信息（更多内容将在以后的各章中介绍），所以每像素所对应的帧缓存位数目可能远超过 24位。例如，对于每个原色可能有一个“屏蔽”位，当该位为1时表示相应像素的这部分是可写或可改变的，当该位为0时表示不允许这些操作。

另一种有助于内存效率的颜色表示，被称为*颜色索引*或*颜色查找表*（CLUTs）。每个像素在帧缓冲区中是固定数量的位（比如说8位）。这时就有一个单独的颜色查找表，该表有2^8=256个入口。每个入口将会包含三个字节，一个字节对应于红色，一个字节对应于绿色，还有一个对应于蓝色。当某个像素将要被扫描和显示时，以它在帧缓冲区中的值（通常被称为*索引*）在查找表中检索，从其查找表的红色、绿色和蓝色字节中得到红色、绿色和蓝色的强度值。这种模式的优点在于帧缓冲区和颜色查找表的内存需求比真彩色模式显著降低。缺点是只有256种不同的颜色能同时显示出来。有一个优点是 CLUT 可以被用于制作简单动画（留给读者作为练习）。将系统设置成允许程序设计者使用真彩色系统或是 CLUT系统，或者让不同的窗口使用不同的方式显示，这些都是可能做到的。在这本书中我们将只使用真彩色系统。 110

从图形程序设计者的观点来看，颜色界面就是一组函数，它们能允许程序员设置像素的红、绿和蓝色的特定组合。这可以通过多种方式来操作。绝大多数的基本函数具有下列类型：

```
SetPixel(x, y, red, green, blue)
```

这里 (x, y) 确定所指的是哪一个像素，x、y为非负整数，坐标范围从 $(0, 0)$ 到 $(M-1, N-1)$。设显示分辨率为 $M \times N$。三个颜色值在范围 0.0~1.0之间，这里 0.0 代表电子束关闭，1.0 代表电子束已经达到它的最大电压值。对在 0.0和1.0 之间值的解释依赖于彩色分辨率

(即依赖于每个像素RGB原色有多少位)。

很重要的一点是要注意到 RGB 值定义了像素控制光束的电压值，而且要注意到在电压和荧光点发光强度之间并不是一种线性关系。事实上这种关系具有形式如下：

$$光强度 I = V^{\gamma} \tag{4-56}$$

这里γ依赖于特殊显示器（一般在约 1.5~3.0的范围内）。为了正确地设定 RGB 值以便得到想要的强度，需要求这个方程的逆：

$$V = I^{\frac{1}{\gamma}} \tag{4-57}$$

所产生的V被用于SetPixel函数。这个过程通常被称为gamma校正。一些显示系统今天仍然在直接执行这个操作。

显示器的 RGB 空间代表了先前小节中描述的颜色空间。因为每个红色、绿色和蓝色的成分都是可以独立地在0.0和1.0之间变化，通常我们都对此给出了示意性的表示，如图4-11 所示。黑色位于点（0，0，0），白色位于点（1，1，1）。连接这两点的直线称为“灰色线”。该图通常叫做“颜色立方体”。然而，它容易造成误解。因为所有 CRT 显示器都是不同的，在一台显示器上得到的颜色，比如（0.5，0.8，1.0），在另外一台显示器上看起来可能就不一样。因此很重要的一点是要回到XYZ 空间工作的问题上，然后将结果转换到特定的显示器上。

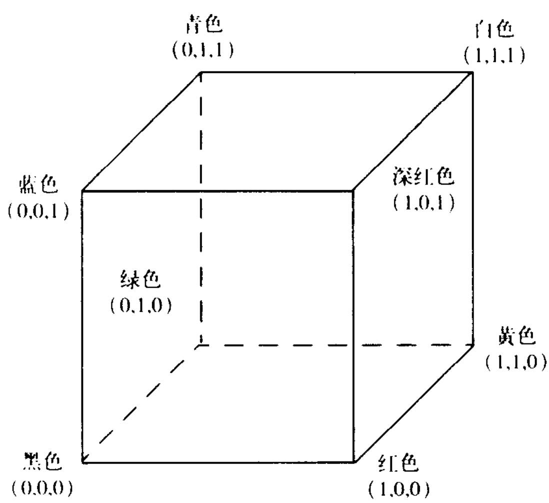


图4-11　RGB颜色立方体：RGB颜色系统的表示

4.9　RGB 和XYZ颜色空间之间的转换

考虑一个显示RGB系统，它有三个原基色 $R(\lambda)$、$G(\lambda)$、$B(\lambda)$。因为它们是实际颜色，会

111 有在XYZ色度空间中的一个表示。假设关系如下：

$$\begin{aligned} R(\lambda) &= X_R X(\lambda) + Y_R Y(\lambda) + Z_R Z(\lambda) \\ G(\lambda) &= X_G X(\lambda) + Y_G Y(\lambda) + Z_G Z(\lambda) \\ B(\lambda) &= X_B X(\lambda) + Y_B Y(\lambda) + Z_B Z(\lambda) \end{aligned} \tag{4-58}$$

因此相应于这些颜色的二维空间XYZ色度坐标是：

$$\left(\frac{X_R}{X_R+Y_R+Z_R},\frac{Y_R}{X_R+Y_R+Z_R}\right)=(x_R,y_R)$$
$$\left(\frac{X_G}{X_G+Y_G+Z_G},\frac{Y_G}{X_G+Y_G+Z_G}\right)=(x_G,y_G) \quad (4\text{-}59)$$
$$\left(\frac{X_B}{X_B+Y_B+Z_B},\frac{Y_B}{X_B+Y_B+Z_B}\right)=(x_B,y_B)$$

实际上，显示设备制造商公布二维空间的色度，因此式（4-59）的右边是已知的。因为在式（4-59）中的分母是未知的，相应于RGB 的XYZ 颜色可以写成：

$$C_R=\alpha_R(x_R,y_R,z_R)$$
$$C_G=\alpha_G(x_G,y_G,z_G) \quad (4\text{-}60)$$
$$C_B=\alpha_B(x_B,y_B,z_B)$$

112

现在我们寻找一个能完成从RGB 到XYZ转换的矩阵A。特别地，RGB颜色（1、0、0）应该映射为C_R，（0、1、0）映射为C_G，且（0、0、1）映射为C_B（这是简单的基的变化）。因此：

$$C_R=(1,0,0)A$$
$$C_G=(0,1,0)A \quad (4\text{-}61)$$
$$C_B=(0,0,1)A$$

所以有：

$$A=\begin{bmatrix}\alpha_R x_R & \alpha_R y_R & \alpha_R z_R\\ \alpha_G x_G & \alpha_G y_G & \alpha_G z_G\\ \alpha_B x_B & \alpha_B y_B & \alpha_B z_B\end{bmatrix} \quad (4\text{-}62)$$

还需要确定（α_R、α_G、α_B）。在 XYZ系统中白色点是$\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$。在RGB系统中白色点一般为（1、1、1）。那么有：

$$(1,1,1)=\frac{1}{3}(1,1,1)A \quad (4\text{-}63)$$

重新整理得：

$$\begin{bmatrix}x_R & x_G & x_B\\ y_R & y_G & y_B\\ z_R & z_G & z_B\end{bmatrix}\begin{bmatrix}\alpha_R\\ \alpha_G\\ \alpha_B\end{bmatrix}=3\begin{bmatrix}1\\1\\1\end{bmatrix} \quad (4\text{-}64)$$

从中可以很容易地求得未知量。实际上，这个矩阵有两个目的：

- 给定在某一特定显示器上的一个颜色，我们希望能在另外一个显示器上重新得到。可以使用矩阵A将它转换到XYZ系统，然后再使用相应于第二个显示器的A^{-1}，将它从XYZ系统转换到第二个显示器的RGB上。
- 给定一个计算得出的 XYZ颜色，我们能够使用A^{-1}转换它到一个特定显示器的 RGB上。这说明了 XYZ 系统的设备独立性质的作用。

4.10 颜色范围和不可显示颜色

CIE-XYZ系统能产生出针对所有可见颜色的条件等色——只不过它的原色是虚构的。因

此一个实际系统不能够建立在这个基础上。另一方面，如我们已经看到的，当原色本身是可见颜色时，它们的线性组合不能够再生出所有可见颜色的条件等色。假设某个显示器具有三原色R、G和B，其相应的XYZ颜色为C_R、C_G、C_B。那么这些颜色的色度将会形成 CIE- XYZ
113 色度图上的一个三角形。这样的一个三角形如图4-12所示。通过围绕式（4-39）的讨论，我们知道在这个三角形里面的所有点都对应于这个系统的一个可见颜色。与一组原色对应的所有这样的可见颜色集合称为该系统的*颜色范围*。显然颜色范围对于不同的显示设备是不同的。图4-12给出了一些例子。那么当一个 XYZ 颜色转换为一个不可显示的等价 RGB 颜色时该怎么办呢？在讨论这个问题之前，先解释几个术语。

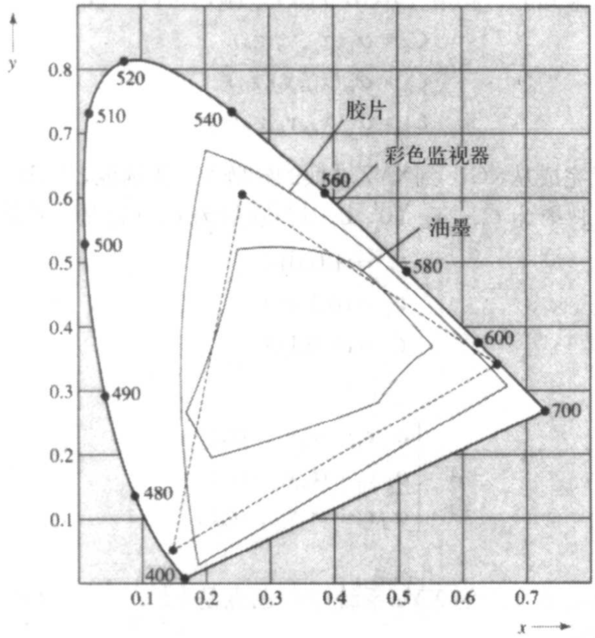


图4-12　多个显示系统的颜色范围

图4-13所示的是一幅CIE-XYZ色度图，特别是位于$\left(\frac{1}{3},\frac{1}{3}\right)$处的白色点W。从图中还可以看到另一个点P。W和P两点的连线与曲线边界相交于点 Q。Q 点对应于一个纯波长颜色，并称其
114 为P点的*主波长*。

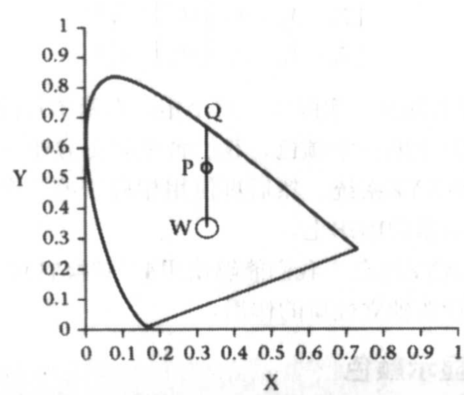


图4-13　CIE-XYZ 给出的白色点W和位于P处的颜色

一种颜色是饱和的，如果它靠近主波长颜色。对饱和度的准确测量可以用一个变量S，公式如下：

$$S=\frac{\mathrm{WP}}{\mathrm{WQ}} \tag{4-65}$$

因此纯颜色是100%饱和的。一种颜色与白色“混合”越多，我们说它越不饱和。由此还可以把颜色看成是三个值的组合：主波长（也叫做色调，是该“颜色”的基本色彩）、饱和度（颜色的“纯”度，表示了该颜色中白色分量的多少，它由光中单色光所占比例决定），以及亮度（这不能在色度图上表现出来的，因为它是与发光亮度无关的）。

现在又假设已经计算出一个XYZ颜色，我们使用式（4-62）的结果将它转换到RGB坐标。转换后的颜色有可能无法显示，这有两个原因（或者是两者同时成立）：

- 它可能落在色度图中三角形的外面，即处于由三原色C_R、C_G、C_B所定义的三角形的外部。标志是至少有一种颜色的强度将是负值。换句话说，这个颜色的色度在所讨论的设备上是不可显示的。
- 另外一种可能性是它的色度是可显示的，但是它所对应的颜色的发光亮度不在这个设备可显示的范围之内。换句话说，假设我们映射该设备的RGB立方体到完全的三维XYZ空间，如图4-14所示。任何落在所映射的立方体外部的颜色，即使它在$X+Y+Z=1$平面上的投影位于色度图之内，也是不能显示的，因为它的发光亮度没有在范围中，此时至少有一种RGB值将超过1。

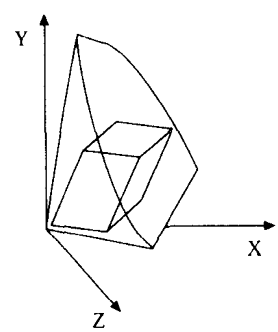


图4-14 将RGB立方体映射到CIE-XYZ颜色空间

对于这种情况如何处理没有一个明确的答案。这个问题称为*颜色裁剪*——如何把一个不可显示的颜色转变到可显示的范围中（深入的讨论可以在（Hall，1989）中找到）。如果颜色落在颜色范围之外，一个策略是降低该颜色的饱和度，沿着PW直线移动直到它进入颜色范围内为止。其优点是保持了颜色的主波长不变。如果它在RGB立方体外部，可以采取的策略是做一个从原点出发经过该颜色相应点的向量，该矢量与RGB立方体相交。如果有这样的一个交点，则新的颜色就在颜色范围中了。它的优点是保持了颜色的色度，但是显然它的发光亮度改变了。对此问题的详细讨论超出了本书的范围。它在颜色精度很重要的一类应用中尤其重要，这类应用如真实感光照等。图形系统一般只是进行简单的修剪处理，方法是对于任何负值将其变为0，对于任何大于1的值将其变为1。另一个策略是求出最大坐标（如果大于1），然后缩放颜色坐标使其最大值为1（留给读者一个问题——在色度上这意味着什么?）。 115

4.11 小结：技术整合

在计算机图形渲染中普遍使用来自实际显示器的RGB空间中的颜色。光源发射器所发出的光是用RGB三元组来表示的（例如一个白色光源表示成（1，1，1）），表面反射函数也是用这些RGB值来表示光能量的，并用这些RGB值来设定像素。同样存在颜色裁剪的问题（如果有多个光源，那么颜色将叠加在一起），但是通常带有负入口的RGB值不会出现。

通过先前的所有讨论我们应该清楚地看到这种方法是相当不正确的。它的错误至少来自两个方面原因。第一是在不同的显示器上进行相同计算将会产生不同的彩色图像。这显然是因为

RGB系统是高度依赖于设备的，而且对于一台显示器的RGB值在另外的一台显示器上可能形成一个不同的颜色。然而，人的视觉系统具有极强的“适应性”，对整幅图像的效果会是相同的。举例来说，在最简单的一种情形中，整个图像比较黑暗，那么这种适应性会使得观察者在两台显示器上看到的是相同的图像（假设观察这两幅图像时有一个很长的时间间隔）。

这种方法不正确的第二个原因是用RGB值来度量光的能量，毫无疑问这是完全不合适的。我们已经了解到一定要区分颜色的两个不同方面：一是能量，它是由物理世界产生的，依据
116 波长分布；二是视觉系统是如何对这种分布做出反应的。RGB系统在这两个方面都没有考虑。RGB 系统是一种描述显示监视器上颜色的方法，它不是描述环境的光能量的方法。我们今天所用到的计算机图形很大一部分都没有区分这些——这是因为这些图形不是很在乎所描写事物的“逼真”程度——也就是说，它不是仿真。

比如会有这样的问题：“所显示的这组颜色对于这个电脑游戏/广告/徽标合适吗？”但这个问题是没有意义的，因为从自然或人的视觉系统观点看，这个问题并没有一个正确的答案——它看起来是否达到了想要传达的效果？——它对于这个游戏（广告、徽标）是增强了其“吸引力”还是降低了这种吸引力？另一方面，当图形有意要仿真环境的光照效果时，就会生成真实的图像，那么RGB系统是完全不适当的。它之所以得到应用是因为在计算上的方便性，但是这种方便性是以牺牲光照仿真的视觉效果为代价的。本书的大部分内容都是有关让图形“看起来像”，而不是有关正确光照的。这不会成为一个问题，但是读者必须意识到使用 RGB 系统的缺陷，它不是表现颜色的正确方法。

那么什么才是更适当的方法呢？图形渲染系统应该计算颜色如$C(\lambda)$的光谱光亮度分布。这意味着所有的光照计算应该对足够数量波长进行计算，以便得到光谱光亮度分布的估计；其次，使用CIE-XYZ 颜色匹配函数将光谱分布转换成 XYZ 坐标：

$$
\begin{aligned}
X &= \int C(\lambda)\bar{x}(\lambda)\,\mathrm{d}\lambda \\
Y &= \int C(\lambda)\bar{y}(\lambda)\,\mathrm{d}\lambda \qquad (4\text{-}66) \\
Z &= \int C(\lambda)\bar{z}(\lambda)\,\mathrm{d}\lambda
\end{aligned}
$$

它们可以被转换成CIE-XYZ色度值，使用式（4-62）所定义的与某种显示器RGB相适应的映射将它们映射为RGB值。RGB 值可能需要裁剪，再经过gamma校正，最后通过函数`SetPixel`或者其他确定一组像素颜色的函数将它们送到显示器上。

这显然要比仅仅使用RGB更复杂，计算强度更大，所以这种方法很少使用就不足为奇了。
117 这个讨论也隐藏了另外的一个重要问题——即$C(\lambda)$是如何获得的？这是一个很困难同时也是很基础的问题——不但需要在足够数量的波长上对分布进行采样，而且这些波长一定要是感知上敏感的波长，这样将会对最后的图像起作用。我们已经了解到视觉系统对蓝色光波长区域的反应要比对绿色和红色等较长波长的光波区域的反应要弱得多，因此一个较好的采样策略应该充分利用这一点。这已经超出了本书的范围——这一方面的最新研究成果请参考 Hall（1999）。

在最近的几章中，我们研究了光以及人对光的反应等一些内容，这个讨论相对抽象，同时也比较深奥。这给我们提供了一些理性认识，但是好像还没有深入到如何在真实的显示器
118 上生成图形这一步。我们在后续的各章中将要具体地介绍这些内容。

第二部分 从真实到实时I

第5章 计算机图形的绘画隐喻

5.1 引言：绘画隐喻

在这一章中我们要提出第一个，也是非常简单的一个关于光亮度方程（式(3-23)）的“解法”。该方法忽略掉方程右边除第一项外的其他项，并且把每个对象本身都看成是一个光发射器，不允许在表面之间的任何相互反射（场景中每个对象的BRDF总是0）。而且，每个表面上所发射的光是完全均匀的——即表面上的任何点处发出的光都是一样的。有关这个解法有很多需要解释的，但是首先我们要对它给出一个具体的讨论，然后给出该方法在渲染方程方面的含义。首先介绍的这个方法称为绘画隐喻。

在图5-1中一个画家正在观察一处景致，并把他所看到的转移到他的画布上。他的技巧在于混合颜料以得到适当的颜色、巧妙的构思和精细的手法。根据这一章的目的，我们对艺术创造进行进一步的抽象，设想画家是一个机器人，它能把在一个固定的观察点处所观察到的景致再现到画布上。

让我们不去考虑观察景物和将所看到的传递到画布上这些问题，这样画家的任务就可以被大大地简化。假设有一块透明的醋酸纤维材料的画布能够遮挡住整幅景致，画家直接在这块画布上作画。让我们进一步假设醋酸纤维材料被矩形网格剖分。画家现在的任务就是连续地观察每个网格单元中的景致，然后选择适当的颜料画在相应的网格上。 119

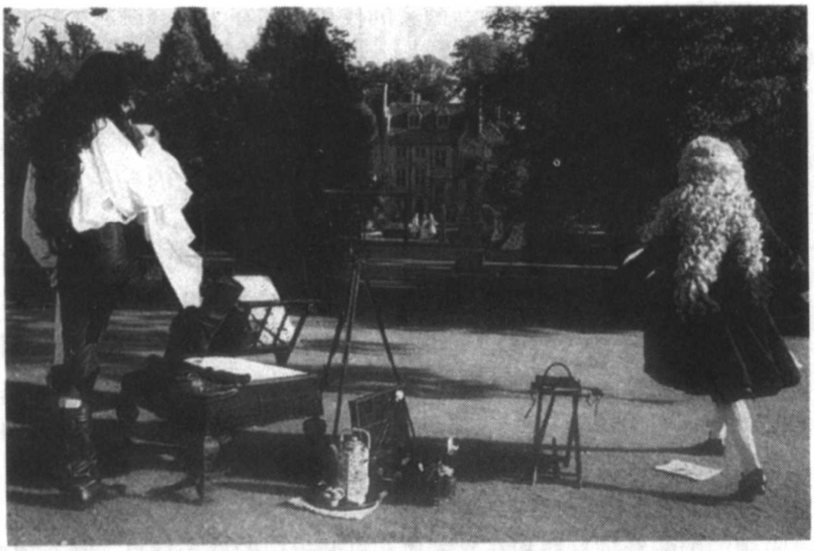


图5-1 画家观察环境并在画布上作画

在这个过程中有许多基本假设。画家的头部位置一定是固定的，每次都从同一个位置看，而且视线必须穿过每个单元的中心点。在图5-2 中我们可以看到，来自场景的光线经过每个单元的中心点汇聚在一个图像点上。这个图像点是“画家的眼睛”。与第 4 章做比较，这是一个
120 经过极端简化了的“眼睛”——事实上，眼睛在这里可以被看成是醋酸纤维画布（类似人眼的视网膜）和单一收敛点（类似人眼的瞳孔)的组合。

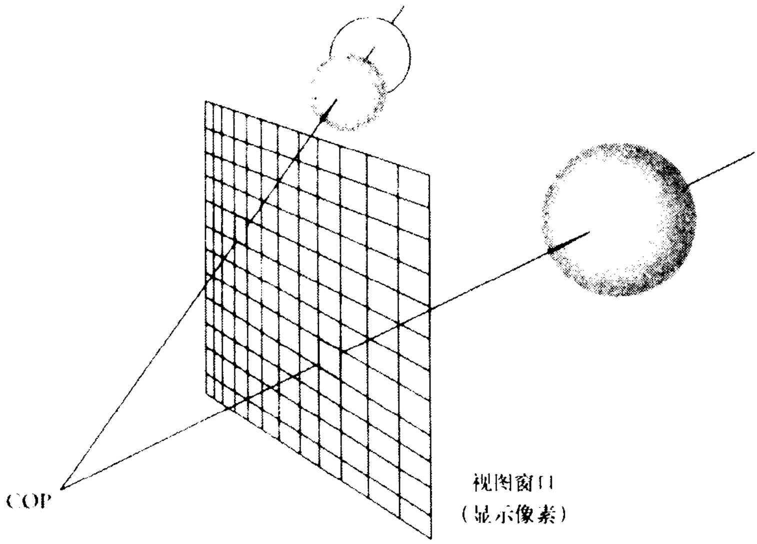


图5-2　光线汇聚在“画家的眼睛”或“投影中心”

我们假设单元足够小，以便穿过每个单元所能看到的只有一种颜色。也就是通过每个单元的中心点所看到的颜色是整个单元颜色的代表。这些假设的一个等价概念是单元可以被看作是无限小、是面积为零的区域——当然，这样的假设与真实的情形相比较显然是不正确的。无论单元如何小，一些单元总会包含场景中的一些边缘以及颜色的陡然变化。然而，我们还是使用单元单一颜色假设作为工作方法。画家的技巧仍然在于混合颜料，在相应单元上用适当的一种颜色来表达从该单元所看到的景色。

这里还有一个更细微的假设。真实的画家，即使是一个机器人，也需要一定的时间来完成一幅图画（尤其当单元无限小，有无穷数目的单元)。在此期间光照条件将会改变，因此颜色将会改变，除非场景是在一个完全封闭的环境中，只有人造光的存在。然而我们假设画家确实是一个机器人，整幅画是瞬时完成的。

事实上，为什么我们要在画家身上费心思？让我们去除掉“人的因素”。图5-3给出了使用旧式的“盒式照相机”如何来产生场景图像的。方盒几乎完全是无光的——除了通过一个很小的光圈外没有别的地方有光进入方盒。光圈位于方盒一侧的中心位置，光线因而打在方盒的另一侧。在这一侧有感光性材料覆盖表面。假设感光材料被分割为一矩形感光性单元阵列。当单元的中心被光线照射时，整个单元根据所接收的光线颜色而“上色”。我们假设整个过程是瞬时的。光圈等价于像点，是“画家的眼睛”。与我们先前的绘画情形相比，重要的区别在于形成在感光材料上的图像是个倒像。

在绘画隐喻中的醋酸纤维材料和在盒式照相机装置中的感光材料，都有图像形成在它们上面。这个图像是通过从三维（现实世界）到二维（胶片或醋酸纤维平面）投影过程来完成

的——光的路线是直线、与一个平面表面相交。当像点位于场景和投影平面之间时所形成的图像是颠倒的（如在盒式照相机的情形）。当像点在投影平面的另一侧时，所形成的图像为正像。 121

在本章的余下部分中，我们从数学角度进一步研究绘画隐喻的模拟，而且使用它来打造计算机图形的基础。

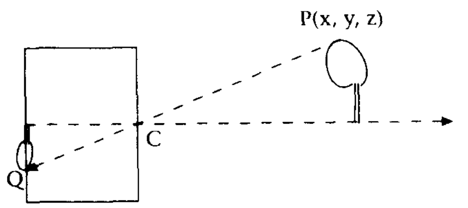


图5-3 盒式或“针孔”照相机。光线从C点进入，每一点（P）被投影到胶片上

5.2 模拟绘画隐喻

术语

画家在醋酸纤维画布上渲染场景的整个景致。在计算机图形中，由醋酸纤维形成的平面被称为**视平面**、**图像平面**或**投影平面**。在本书中我们一般使用“视平面”这个术语。视平面是一个平面，该平面是一个无限平面。然而，画家只能看到视平面的一个特别矩形部分，这个矩形部分就是画面所在的部分。我们称该画面为**视平面窗口**。画家通过视平面窗口观察场景并渲染它。视平面窗口被剖分为矩形网格单元。每个单元称为一个**场景像素**。场景像素具有有限区域。赋予每个场景像素的颜色就是穿过该像素中心点所看到的颜色（这个假设将会在稍后被放松）。画家的眼睛就是来自场景的光穿过像素的中心点后的汇聚点。该汇聚点有各种不同的叫法，可以称之为**视点**、**像点**或**投影中心**（COP）。本书比较倾向于使用最后的这个术语。

绘画隐喻有三个主要的成分需要被进一步研究：场景、视图和渲染过程。我们将依次讨论这些概念。

场景

场景是对象的一个集合。每个对象有它的几何和材质属性。对象的形状是由它的几何属性所确定。举例来说，它可能是球体、立方体、四面体，或者是某种多面体，甚至仅仅是一个平面多边形，比如一个三角形。在这一章中我们将用一个球体作为主要的例子，并假设场景只由球体所构成。

每个对象有材质属性。在计算机图形学中，我们主要感兴趣的是那些决定对象如何反射入射光的材质属性，甚至是否对象本身就是一个光发射器。这里我们再一次做一个极端的简化假设，假设每个对象（都是球体）具有一个固定的“颜色”。每个球体由它的中心点和半径所决定。这组球体所形成的场景位于一个三维空间中，该三维空间采用右手坐标系来描述，如图2-1所示。 122

描述整个场景的坐标系称为**世界坐标系**（WC）。暂时我们可以认为该坐标系的选取是任意的，由场景设计者根据喜好随意设定。

视图：简单照相机

视图是“观察”场景的方式。在这一章中我们对视图给出一种更严格的定义，称之为**简单虚拟照相机**（或**简单照相机**）。投影中心是位于Z轴正方向上某处的一个点。视平面距离是

自原点到COP点的距离。视平面（醋酸纤维平面）是XY平面。视平面窗口因此是 XY 平面上的一个矩形区域。视平面和COP点是不能充分决定投影的。从盒式照相机类比中我们可以看到COP点可能位于视平面的任一侧（如果它是在视平面的“后面”，那么图像向会是正向的，否则它将会是反向的）。因此还需要另外的一个参数来决定视图的主方向，这被称为视平面法向。它是视平面的法向，与从COP点出发射向场景像素的任何光线构成锐角。它是视平面的“前面”。

在上面的假设中，我们把视平面法向与负Z轴方向对齐。既然我们限制了COP位于正 Z 轴上，而且视图的方向沿着负Z轴的，所形成的图像将会是在XY上的一个正像（在视平面窗口矩形范围内），假若所有的场景对象都位于Z轴的反方向上。

因此，一个视图（此时视图被描述为简单照相机模式）是由下列各项参数来描述的：COP、视平面窗口（VPW）和视平面法向（VPN）。这些参数由图5-4说明。

123

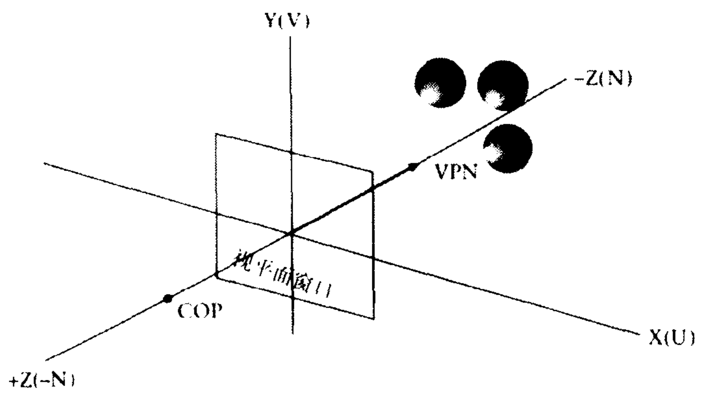


图5-4 用XYZ世界坐标表示的简单照相机

假如我们重新标记轴，将正X轴定义为正U轴，将正Y轴定义为正V轴，并且将负 Z 轴定义为正N轴。所构成的UVN系统称为左手坐标系，通常叫做**观察坐标系**或**眼睛坐标系**。它是从眼睛的角度来描述场景的，而不是从外部画面的角度来描述，这也在图5-4中示出。需要注意的是观察坐标系与世界坐标系具有完全相同的标尺，只是方向不同。我们在稍后的章节中要学习的内容之一就是该如何在观察坐标系（VC）中重新描述场景，就像世界坐标系（WC）那样。对于简单照相机模式，这是一件轻松的任务。

理解简单照相机模式的最好方法就是让我们自己设计一个虚拟的世界。我们可以在地球之外的“其他世界”（比如在火星上）中架设一部观察设备（例如摄像机），并从我们“自己的世界”来控制它。这里我们构造一个虚拟的世界，简单照相机就是我们在这个虚拟世界中所构造并控制的第一个照相机（当然它是很简单的）。正如我们把摄像机转接到一个显示器，并能通过显示器观看摄像机所“看见”的景物那样，我们要将简单照相机转接到显示器或是其他显示设备上，从而可以看到虚拟的世界。但是如何实现从简单照相机摄制的图像到显示器显示的转接呢？

渲染过程：光线投射

给定一个场景和一个视图，我们能设计一个方法来“渲染场景”，这等价于画家透过每个

单元的中心点观察景物，确定场景的颜色，并用颜色来填充该单元这样一个过程。

我们所使用的方法是基于亚里士多德对人的视觉认识的一种简单实现。在这个理论中，能量是从眼睛送出的。当能量到达世界中对象时，视觉就产生了。它引起眼睛和大脑的感知，这是视觉的基础。这个概念是与我们在第1~4章中所理解的概念相反的。

然而，跟随亚里士多德的概念还是有一些优点的——它的算法非常简单，而且直接导致了许多在当今计算机图形学中普遍使用的主要概念的产生。事实上，它导致了光线跟踪方法的形成，这为高度相片逼真的图像的生成奠定了基础。我们将会在今后对此做进一步的分析。首先考虑光线跟踪的一个简化做法，叫做**光线投射**。

光线投射的含义是沿着光的路线，从投影中心开始穿过场景像素的中心点。每条光线代表了进入"眼睛"的光的"反"方向。对任何一束这种光线，求出它与场景中对象的第一个交点（如果有的话），依照所相交的那个对象的颜色对场景像素着色。 124

执行过程是对每条从COP点开始并穿过场景像素中心点的光线进行的。在这个过程的最后，场景像素将会在视平面窗口上形成一个图像。这是与画家在醋酸纤维画布上作画一一类比的。我们现在从数学的角度来分析一下这个过程。

视平面窗口被分割成$N \times M$个屏幕像素。坐标为（0，0）的像素位于视平面窗口的左下角，坐标为（$M-1$，$N-1$）的像素位于右上角。设视窗口中x坐标的最小值为$xmin$，最大值为$xmax$，y坐标的最小值为$ymin$，最大值为$ymax$，则视窗口左下角处的坐标为（$xmin$，$ymin$），右上角的坐标为（$xmax$，$ymax$），每个像素的z坐标值为0，因为视平面位于XY平面上。

与每个像素（i，j）对应的是一个矩形区域。每个单元的宽度为：$width = \frac{xmax - xmin}{M}$，高度为：$height = \frac{ymax - ymin}{N}$。因此对于像素（$i$，$j$），它的左下角坐标为：（$xmin + width \times i$，$ymin + height \times j$），它的右上角坐标为($xmin + width \times (i+1)$)，$ymin + height \times (j+1)$)，这里$i=0, 1, \cdots, M-1$；$j=0, 1, \cdots, N-1$。因此，标记为$(i, j)$像素的中心点在$\left(xmin + width \times \left(i + \frac{1}{2}\right), ymin + height \times \left(j + \frac{1}{2}\right), 0\right)$

给定两点p_0、p_1，从p_0点到p_1点的方向矢量是$dp = p_1 - p_0$。从p_0开始方向为dp的光线的参数化方程为：$p(t) = p_0 + t\,dp$，$t \geqslant 0$。如果我们将p_0点作为COP点，它的三维坐标为（0，0，d），这里d为视平面的距离，将p_1点设为像素点（i，j）的中心点，那么上面的参数化方程就是经过这个像素的光线的方程。

球心位于坐标原点，半径为r的球体方程是：

$$x^2+y^2+z^2=r^2 \tag{5-1}$$

在球体和光线相交处，球体方程和光线方程都要同时被满足。此时有：

$$x(t)^2+y(t)^2+z(t)^2=r^2$$

这里我们使用记号如下：

$$p(t) = (x(t), y(t), z(t)) = (x_0 + t\,dx, y_0 + t\,dy, z_0 + t\,dz)$$

将它们代入球体方程，得到关于参数t的二次方程：

$$t^2(dx^2 + dy^2 + dz^2) + 2t(x_0\,dx + y_0\,dy + z_0\,dz) + (x^2 + y^2 + z^2 - r^2) = 0$$

该方程具有如下形式：

$$At^2+2Bt+C=0$$

其解为：

125

$$t=\frac{-B\pm\sqrt{B^2-AC}}{A}$$

如果判别式$B^2-AC<0$，那么光线不与球体相交。如果判别式等于0，则光线与球体相切，否则它们有两个交点。在这种情况，最近的一个交点才是我们所感兴趣的，相应解为：

$$t=\frac{-B-\sqrt{B^2-AC}}{A}$$

这提供了求解位于坐标原点的一个球体与光线相交问题的方法。如果球体不在坐标原点，该解法仍然可用。对于位于坐标（a，b，c）的球体的一般方程如下式，可以直接用相应参数代入上面公式求得交点：

$$(x-a)^2+(y-b)^2+(z-c)^2=r^2 \tag{5-2}$$

然而，假设我们所写的是针对位于坐标原点的球体与光线相交的程序。我们可以重复使用这个程序来求解更一般的问题。如果我们平移球体使得新的中心位于坐标原点，同时平移光线相同的量，那么新的光线将会与新球体相交，相交处的t值与原先的光线和球体相交处相应的t值是完全相同的。新的光线的原点为$p_0-(a, b, c)$，它的方向矢量没变。

在这个渲染过程中，我们需要求出光线与最靠近的球体相交处的t值。这需要求出所有球体与光线的交点，并记录下所发现的最小t值（如果有相交的话）。场景像素就设定为对应的球体颜色。如果光线没有与任何球体相交，那么场景像素应该设定为其默认值（或许为“黑”色）。当然，这一过程包括对所有从COP点开始经过场景像素中心点的光线执行上述操作。

形成显示图像

上面描述的渲染过程在计算上是非常简单的，尽管很费时。设$r(i, j)$是从投影中心开始经过标记为(i, j)场景像素的光线。那么对于每个$i=0, \cdots, M-1$和$j=0, \cdots, N-1$，我们必须让场景中每个球体都与光线$r(i, j)$相交，以便求得最小的t值。所对应球体（如果有的话）的颜色就设为标记为(i, j)的场景像素的颜色。有很多方法可以加速这个过程，我们准备在光线跟踪的部分对此进行讨论。

设想在计算机程序中已经实现了上面的各项内容。我们会在显示屏幕上看到什么呢？回
126 答应是“什么也没有！”——不会有图像在显示器上形成。场景像素不是屏幕像素。我们必须在场景像素和真实显示像素之间建立对应关系，以便对场景像素颜色的设定变成对真实像素颜色的设定。

首先让我们考虑更一般的问题。假如有两矩形窗口$[xmin, xmax]\times[ymin, ymax]$和$[vxmin, vxmax]\times[vymin, vymax]$。第一个区域被指定为窗口，第二个被指定为对应的视口。具体地说，第一个代表了视平面窗口，而第二个表示一个抽象矩形，我们将把窗口映射到该显示区域上（之所以是一个抽象表示，是因为我们暂时假设它是一个连续空间，而事实上显示屏幕是不连续的）。

这个映射应该具有什么样的性质呢？首先，它应该保持线性。换句话说，直线应该被映射

为直线。其次，窗口中的点应该被映射为在视口中对应的点——举例来说，拐角处应该映射为对应的拐角处。第三，x和y成分的映射应该是独立的，比如在窗口中平行的直线被映射为视口中相当的平行直线。

考虑到这些条件，映射应该具有形式：

$$x_v = A + Bx$$
$$y_v = C + Dy$$

这里（x_v、y_v）在视口中相应于窗口中（x、y）的那个点。根据拐角一定会映射为拐角这个条件，常数A、B、C和D可以被求出，从而得到映射为：

$$\begin{aligned} x_v &= vxmin + \frac{\mathrm{d}v_x}{\mathrm{d}w_x}(x - xmin) \\ y_v &= vymin + \frac{\mathrm{d}v_y}{\mathrm{d}w_y}(y - ymin) \end{aligned} \tag{5-3}$$

这里$\mathrm{d}v_x$和$\mathrm{d}w_x$分别是视口和窗口的宽度，对y有相似的描述。

这个映射可以使用在光线投射的情形中。然而，我们需要将场景像素的中心映射为真实的显示像素。假设在显示屏幕上我们建立了一个窗口，所具有的像素与场景像素数相同。因此这个显示窗口画面的宽度为M、高度为N。从视平面窗口到显示窗口映射中相应点为：

左下角：

$$\left(xmin + \frac{width}{2}, ymin + \frac{height}{2}\right) \to (0,0)$$

右上角： 127

$$\left(xmin + width \times \left(M - \frac{1}{2}\right), ymin + height \times \left(N - \frac{1}{2}\right)\right) \to (M-1, N-1)$$

5.3 图形的主要概念

伴随着这一章中给出的基本隐喻，有一些计算机图形学的基本概念需要介绍。

场景描述、观察和渲染。有三个主要过程必须考虑。第一是场景本身的构造。在我们的例子中，这一点是非常简单的——只是摆放了一些球体。每个球体只有两个参数——球心和半径。在三维空间中摆放一组球体构成所需要的场景相对来讲比较简单。在第8章中我们将会考虑场景建模和构造的一般过程。

第二是视图的描述。注意到这是与场景无关的，因为当我们构造场景时是不考虑如何建立视图的。对于同一个场景可能有无穷多的可能视图。通常一个照相机应该允许从任何一点、在任何方位和任何方向上观察场景。这对于简单照相机是不可能做到的，但是即使使用简单照相机，也还是有无限多可能的视图可供选择——举例来说，通过改变视平面窗口和COP的参数。

第三是渲染方法。在这一章中我们只考虑一种渲染方法，这就是基于光线投射的渲染。通常在给定一个场景和对视图的定义后，会有很多可能的渲染方法可以使用——最终决定于应用的目的。

视景体。通过COP点和视平面窗口的所有光线的集合在场景空间中构成了一个金字塔形

的体。只有与这个体相交的对象在最后形成的图像中才有可能是可见的，我们将这个体称为视景体。当然，即使一个对象与这个视景体有相交，它还可能被其他的对象遮挡，所以它也有可能是不可见的。因此视景体是由COP和视平面窗口隐含确定的。用于描述视景体的参数通常有两个。它们是近裁剪平面和远裁剪平面，如图5-5所示。它们都是与视平面平行的平面，近裁剪平面离COP点较近、远裁剪平面离COP点较远，在近裁剪平面之前和远裁剪平面之后都没有任何对象能投影进入视平面。

将最后图像限制为只有在视景体内的那部分，我们将上述限制称为裁剪。这个视景体（或称为裁剪体）定义为一个六面体——每个面分别
128 称为上平面、左平面、右平面、下平面以及前平面和后平面。前四个平面由光线投射过程隐含给出（在上下平面和左右平面之外的光线都不会生成，因为只有在视平面窗口内的光线才被考虑）。关于近平面和远平面，我们实际上所关心的是由近平面和远平面所截得的那段光线。

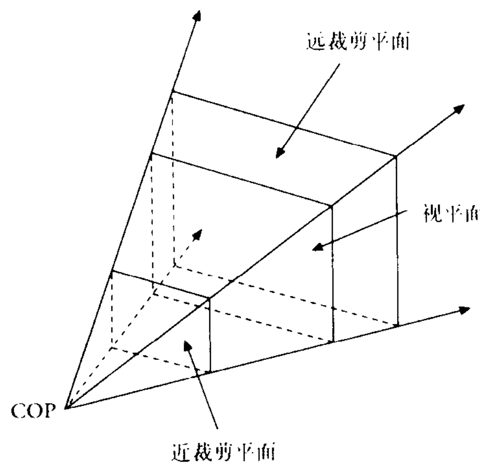


图5-5　视景体

在稍后的时间里，我们将考察渲染的不同方法，我们将会执行一个裁剪过程，以便在渲染过程开始之前删除视景体之外的场景部分。

走样。最后图像的质量取决于所采用的视平面窗口的分辨率（该分辨率应该是与最后显示屏幕的分辨率一样）。彩图5-6给出了一个图像序列，它们的分辨率依次逐渐增加。然而，不只是分辨率决定图像的质量。渲染过程的本质是对连续现象的离散采样。可用的颜色分辨率是有限的。场景像素和显示像素都是一个有限的数量。与每个场景像素相对应的只有一个颜色，但是场景像素本身是个面积非零的区域，所以有许多场景中的颜色会经过它被观察到。这个离散化在最后图像中产生了所谓的走样现象。这种现象的产生有两方面的原因。一是由于对场景中连续实体采样的频率较低造成的（平滑的曲线和直线产生了锯齿形）。二是由于用单一颜色来表示多种颜色造成的（当场景中的边界是由两种颜色所构成，该边界正好落在场景像素相对应的体积内）。无论对于上面任何一种对连续实体进行离散表示的情形，降低走样的一个可行的方法就是提高采样频率。对于每个场景像素，我们先前使用一条经过像素中心点的光线。取代方案是经过每个场景像素发射很多条光线，并对每条光线所求得的颜色取平均值。处理这个问题的另一个方法是允许场景像素的分辨率比显示像素分辨率高。那么多个场景像素可以映射到一个显示像素上。然而，
129 维持场景像素和显示像素之间的对应在放宽，今后我们可以自由地选择在场景像素上的任何类型的采样模式。举例来说，我们可以让光线穿过每个场景像素的顶角或是中心点，可以对每个像素用规则网格采样、用不规则网格采样（一个抖动的采样模式）或甚至使用在每个场景像素里面的随机点进行采样。

这样，每个场景像素会有许多光线穿过它，像素的最终颜色确定为由每条光线产生的颜色的一个平均值。注意这个平均值不一定是一个简单的平均值，而是一个加权平均值，依照光线在每个场景像素里面的位置做加权处理——对于靠近中心的位置权值较大，周边位置处的权值较小。

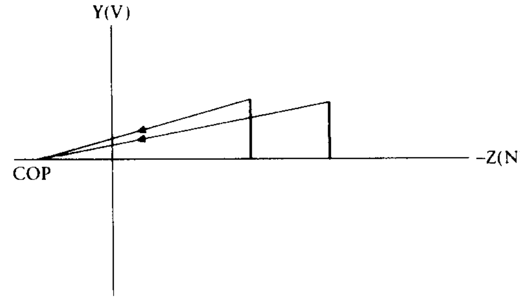


图5-7　图像的尺寸取决于到COP的距离

投影。渲染过程包括从 3D 到2D的“投影”。球体是在3D场景空间中的三维对象。图像平面是两维的。光线投射方法隐含地执行了从3D到2D投影的任务。所实现的投影类型被称为是“透视”投影，如果这个操作过程如自然视觉的模式（但是不完全相同）。在透视投影中，具有相同尺寸的对象投影后的大小取决于它们离COP的距离。如果距离COP较近，则投影对象较大，否则较小，这里假设COP是固定的，如图5-7所示。较近的对象比较远的对象显得大，尽管它们可能具有相同的尺寸。在稍后的一章中我们将会看到该如何明确地计算一个透视投影，以及它的一些性质。

光照。在光线投射到对象上进而求得颜色的过程中我们做了一个很大的简化。这些颜色是从哪里来的呢?在对应的绘画隐喻中颜色是由所参与的表面材质属性之间的交互形成的（它们是如何发出光、如何反射光和如何吸收光）。一些对象是光发射器，一些光只是反射光，而有的对象既能发射光同时又能反射光。这种光照本身必须是确定的，理想情况每条光线都有 130
一定的光谱分布。显然，用一种单一颜色来描述每个对象是不正确的——事实上，在通过本章相关的程序所生成的图像中，我们将每个对象当作了一个光发射器。图像看起来像是个平坦的盘子（事实上是个椭圆）。投影的球体看起来不像是球体，它们没有给人一种三维的印象。我们将在下一章中给出关于光照计算的方法，让对象有反射光的能力。

光亮度方程

绘画隐喻如何与光亮度方程的解法发生联系的呢？如我们在本章引言部分中所提到的那样，这个解法等价于将每个 BRDF 看作为 0，因此方程简化为:

$$L(p,\omega)=L_e(p,\omega) \tag{5-4}$$

解法的第二个方面是只对方程的一个非常有限的变量集合求解。p点是COP，方向集合限制为经过场景像素的中心点进入COP的光线方向的范围。该方法完全如式（3-23）紧接着的那段讨论中所描述的那样——每条光线沿着这些方向被反向跟踪（从场景中射出），直到它与某个对象相交（如果有的话）。因为每个对象表面所发射的光线是个常数，与这个光发射相对应的“颜色”是很容易从表中查出来的，并把颜色赋予图像像素。注意，在这个方法中我们一直都是在 RGB 显示器颜色空间中讨论问题的——这是一个更大的简化假设，如同我们在第4章中所看到的那样。

5.4　小结

图5-8 说明了在这一章中所介绍的各个不同的概念。基本思想是场景是由对象所构成的。对象具有几何和材质两方面属性。我们作了一个简化假设，即场景中惟一的几何形状就是球 131
体。材质属性只包含对象的颜色，以RGB格式表现。

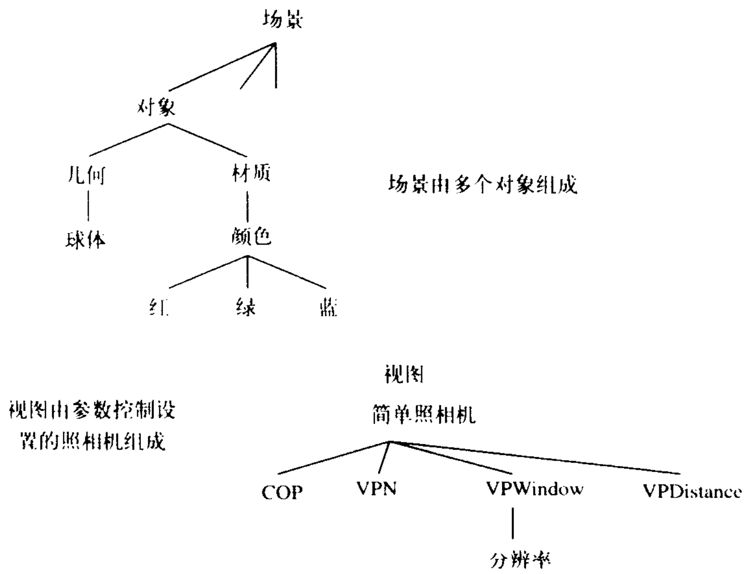


图5-8　场景和视图的结构

场景的视图是由简单照相机的参数所决定的。投影中心（COP）是经过视平面窗口的所有光线汇聚点。在简单照相机模式中，COP被限制在正Z轴上，视口的方向即视平面的法向，为负Z轴的方向。视平面本身是XY平面，而且视平面窗口为XY平面上的矩形区域——相互之间轴线对齐的一些矩形区域。

每个主要组件（场景、对象、材质、颜色和简单照相机）都可以表示为程序中的类，直
132 接由概念映射到类结构。

在下一章中我们将提高图像的真实感，这可以通过增加对象材质属性方面的考虑以及添加光源到场景中来实现。

第6章 局部光照和光线跟踪

6.1 引言

这是最后一章，用来集中阐述有关描述简单场景的几何和观察方面的内容。在这一章中我们把对象视为反射光的材质实体，因而被感知为具有一定的颜色。为了要做到这一点，我们专注于对象的**表面**，而非对象体。感知心理学家J.J. Gibson（Gibson，1986）指出：表面是物质三态中任意两态之间的一个**界面**，所谓的物质三态指的是固态、液态和气态。举例来说，在一种媒介（例如空气等周围媒介）和组成对象的物质（如金属）之间。

在计算机图形学中我们考虑一种极端情形，即把材质看成具有两种光反射类型的表面：**完全漫反射**和**完全镜面反射**表面。假设一光线照射到表面上的某一点，漫反射表面所散射的光位于以表面上这一点为中心的一个半球内，在该半球内所有方向上光是均匀发散的。镜面 133
反射表面（对于透明表面），其反射光的方向位于一个很小的立体角范围内，该立体角决定于入射光的方向和表面的法向。**完全镜面**表面上的反射光具有惟一的方向，而不是一束窄带光束。由此可见在环境中表面之间光的传播至少有四种（理想）类型，如图6-1所示（Wallace et al.，1987）。这里有两个表面，一个表面接收并反射来自外部的光线，一个表面接收来自前一个表面的反射光并将该光线反射出去，这里有四种情形。通常**光线跟踪**就是对图中a所示情形的一个很好的建模，这里的所有表面都是完全镜面反射器和发射器。所谓的**辐射度**模型很好地描述了图中d所示的情形，这里所有表面都是理想的漫反射器。b和c所示的情形依赖于蒙特卡洛方法，将在第22章中讨论。

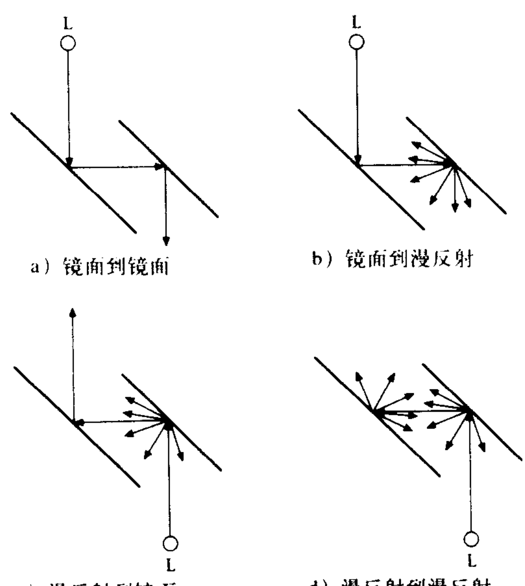


图6-1 光传播的四种机制

在这一章中我们首先考虑基本的局部光照模型，它给出了光源和单一表面之间交互的合理模型。这个模型自从19世纪70年代初就开始在计算机图形学中使用了。它假设光源是单一点，或者描述为从无限远处某点出发的一组方向矢量。它是一个局部模型，因为它只处理光
134 源和表面之间的交互，不考虑图6-1中所示的在表面之间的光反射（除非使用某种近似方法）。基于这种局部光照模型的明暗处理计算是简单的，对于简单场景给出的结果是可接受的，对于实时 3D 动画和交互是足够快速的。图形工作站对 3D 明暗处理所提供的硬件支持一般就是使用这些模型的一个子集。

我们继续介绍在先前一章中已经介绍过的渲染方法，这里我们进一步展开，把它作为光线跟踪的入门材料。这是一种全局光照方法，要解决如图6-1a中多个镜面反射面的问题。

6.2 漫反射和朗伯定律

漫反射发生在完全不光滑表面，这里入射光的反射从任何角度看都似乎是“一样的”。更严格地讲，这种表面遵从朗伯余弦定律。朗伯定律指出：对于漫反射体，表面上一点处任意方向的反射光强度和光源入射角（入射光线和表面法向量的夹角）的余弦成正比。因而，漫反射表面那一点上的光亮度同样与观察角度无关。

如果设I是光强度，那么有：

$$I \propto \cos\theta \tag{6-1}$$

这可由图6-2说明。这里矢量V朝向视点。然而，由式（3-14）我们可以得到：

$$L \propto \frac{I}{\cos\theta} \tag{6-2}$$

这里L是光亮度。所以对于这样的表面，光亮度是独立于观察角度的。对于从一完全漫反射（或朗伯）反射器上的一点所发出的光，光亮度（因此也是亮度）都是一样的。

让我们回想一下式（3-18），该公式说明入射光在某个方向上的反射光的光亮度等于辐照度（irradiance）乘以BRDF。考虑图6-3中所示的情形，光线来自一个光源面片(S)，照射到接收表面(R)，并从R反射出去。

根据式（3-9），我们得到从光源发出的到达接收表面的辐射能量是：

135 $$\Phi = L_i \cdot \mathrm{d}S \cdot \cos\theta_S \cdot \mathrm{d}\omega \tag{6-3}$$

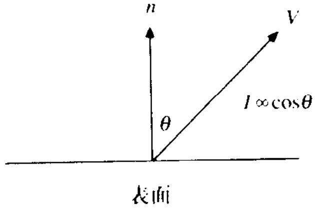


图6-2 朗伯余弦定律

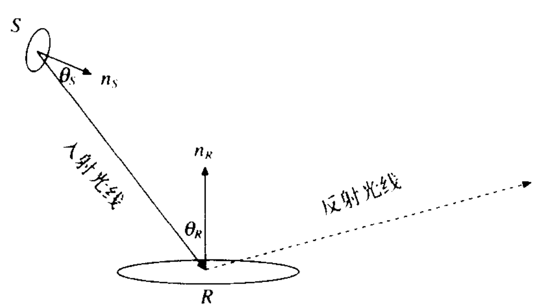


图6-3 从S到R的辐射能量

这里L_i是沿着光线的入射光亮度，θ_S是光源面片的法向和光线方向之间的夹角。展开微分立体角：

$$\Phi = L_i \cdot \cos\theta_S \cdot \mathrm{d}S \cdot \frac{\mathrm{d}R \cdot \cos\theta_R}{r^2} \tag{6-4}$$

这里r是两面片之间的距离。辐照度（E）是接收表面每单位面积上的辐射能量，因此有：

$$E = I_i \cdot \frac{\cos\theta_R}{r^2} \tag{6-5}$$

再一次使用式（3-14），这里I_i是入射光强度。既然从这一点反射的每一条光线的光亮度都是一样的，BRDF 也一定是个常数，由式（3-19），得：

$$L_r = \frac{k_d}{\pi} \cdot I_i \cdot \frac{\cos\theta_R}{r^2} \tag{6-6}$$

这里 L_r是反射光亮度，k_d是**漫反射系数**。它是0和1之间的值，为从表面反射的光所占的比例（另一部分为表面所吸收的成分）。注意它是与波长相关的。现在我们可以看到，漫反射的朗伯定律的另外一个形式是：任何反射光线的光亮度正比于入射光的强度和一个角度余弦的乘积，该角度余弦指的是表面上反射点处的法向与入射光线的夹角余弦。

现在考虑在计算机图形学的局部光照模型中一般是如何处理这些关系的。由式（3-14），我们有：

$$I_i = L_i \cdot \mathrm{d}S \cdot \cos\theta_S \tag{6-7}$$

当其他项不变，这个量将在θ_S=0处达到最大值。我们把它写成：

$$I_{i,\max} = L_i \cdot \mathrm{d}S \tag{6-8}$$ 136

同样地，L_r将在θ_R =0和 k_d=1处达到最大值。同时使用式（6-8）有：

$$L_{r,\max} = \frac{1}{\pi} \cdot \frac{I_{i,\max}}{r^2} \tag{6-9}$$

使用式（6-6）和式（6-9）有：

$$\frac{L_r}{L_{r,\max}} = k_d \cdot \left(\frac{I_i}{I_{i,\max}}\right) \cdot \cos\theta_R \tag{6-10}$$

光亮度的比率与光强度的比率是相等的，因此可以写成：

$$\frac{I_r}{I_{r,\max}} = k_d \cdot \left(\frac{I_i}{I_{i,\max}}\right) \cdot \cos\theta_R \tag{6-11}$$

我们称光强度与其最大值的比为**规范化强度**，写成：

$$\bar{I} = \frac{I}{I_{\max}} \tag{6-12}$$

那么有：

$$\bar{I}_r = k_d \cdot \bar{I}_i \cdot \cos\theta_R \tag{6-13}$$

规范化强度值在0和1之间。通常计算机图形学总是使用这些规范化强度进行相关计算，并把它们映射到显示器的 RGB 值。因此式（6-13）需要计算三遍，分别对红色、绿色和蓝色强度各计算一次。从现在开始，我们要去掉I变量上的横杠，除非特别声明，假设其值为规范化强度。因此方程的最后形式是：

$$I_r(\lambda)=k_d(\lambda)\cdot I_i(\lambda)\cdot\cos\theta_R \tag{6-14}$$
$$\lambda=\lambda_{Red},\lambda_{Green},\lambda_{Blue}$$

这里λ_{Red}、λ_{Green}、λ_{Blue}是显示器主颜色的强度。例如，对于白色光源，有：

137 $$I_i(\lambda_{Red})=I_i(\lambda_{Green})=I_i(\lambda_{Blue})=1 \tag{6-15}$$

如果对象颜色发红，那么$k_d(\lambda_{Red})=k_{d,red}$值会选择接近于1.0，同时$k_d(\lambda_{Green})=k_{d,green}$和$k_d(\lambda_{Blue})=k_{d,blue}$取值接近0.0。

对象可能直接发出光（例如灯泡在有电流通过时），有些对象只能反射光。前者被称为*光源*或*发射器*。在本章的其余部分我们始终假设光源都为*点光源*。因此，光由点来表示它在空间中的位置，用一个规范化的强度来定义它的发射光。

6.3 计算局部漫反射

图6-4 给出了在表面上的一点的法向（N）和从该点到光源的方向矢量(L)。根据朗伯定律，在一点处的反射光强度正比于两矢量N和L夹角的余弦。由于对于完全漫反射模型来说，光在任何方向上的散射都是一样的，观察者的位置可以不用考虑。

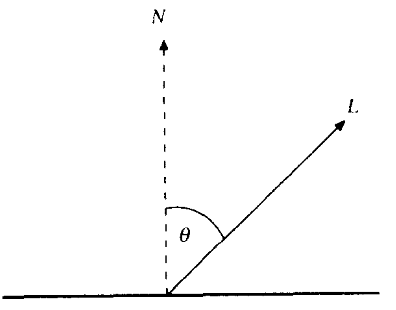


图6-4　朗伯定律：反射强度正比于cosθ，L指向光源的方向，N是表面法向

设矢量N对应的规范化矢量为n，矢量L对应的规范化矢量为l，则朗伯定律可表示为：

$$I_r\propto n\cdot l \tag{6-16}$$

然而，光反射的实际量依赖于材质的表面属性——特别是材质吸收了多少入射光。而且，材质对于每一种波长的光的吸收程度是不一样的。如我们所看到的，漫反射系数定义了每个波长的入射光能量反射进环境的比例。

考虑到漫反射系数，式（6-16）变成：

138 $$I_r=k_dI_i(n\cdot l) \tag{6-17}$$

这里I是来自光源光线的规范化强度。

式（6-17）是相当简单的。这个模型指出从表面上点所反射的光能量的强度由三个因素决定：

- 表面入射光的规范化强度值决定于点光源所发射的光线（I_i）；
- 反射光决定于朗伯定律的作用——即依赖于表面上光线的入射角度（θ）；
- 相对于被材质吸收的光，该材质表面反射光所占的比例（k_d）。

有一个成分通常被加进这一模型。这就是所谓的*环境光*，它是一个具有固定强度的光照，来自环境反射，假定入射光均匀地从周围环境中入射至景物的表面，并等量地向各个方向反射出去，存在和遍布于场景各处（独立于特殊光源）。如果I_a是环境光的总量，那么

$$k_aI_a \tag{6-18}$$

是一特别表面上反射的环境光数量，这里k_a为该表面的*环境反射系数*。

既然局部模型不考虑光在表面之间的传播，这个环境项就可以看成是对环境中所有的交

义反射光（不包括直接来自发射器的光）的一种全局近似。

现在将式（6-16）、式（6-17）和式（6-18）结合在一起，我们获得了完全漫反射器对于单一点光源的局部模型：

$$I_r = k_a I_a + k_d I_i (n \cdot l) \tag{6-19}$$

一定记得这是要计算三遍的，分别为红色、绿色和蓝色三种主色各计算一次。因此式（6-19）的计算如下：

$$\begin{aligned} I_{r,red} &= k_{a,red} I_{a,red} + k_{d,red} I_{i,red} (n \cdot l) \\ I_{r,green} &= k_{a,green} I_{a,green} + k_{d,green} I_{i,green} (n \cdot l) \\ I_{r,blue} &= k_{a,blue} I_{a,blue} + k_{d,blue} I_{i,blue} (n \cdot l) \end{aligned} \tag{6-20}$$

系数k、入射光强度和环境光强度都是由场景设计者选择的，所计算的反射强度被直接用于设定显示器的RGB颜色。

需要强调一下的是，从自然角度来看，这个模型是错误的，从前面一章中我们就能清楚地看到这一点。然而，它所给出的结果是足可以接受的，现在数十亿美元的产业基于这个模型，生产出专门硬件来支持它。

如果有多个光源（设光源数为M），式（6-19）变成：

$$I_r = k_a I_a + k_d \sum_{j=1}^{M} I_{i,j} (n \cdot l_j) \tag{6-21}$$

这里$I_{i,j}$是第j个光源的强度；l_j为第j个光源的规范化方向矢量。

从实际应用的角度，需要注意式（6-21）的计算结果对于任何特别的颜色可能都是在范围[0，1]外的光强度，所以颜色裁剪是需要的，正如在先前的讨论中所说的那样。同时要注意到，$n \cdot l<0$说明在表面的后面有一个光源，所以$n \cdot l$总是被削减为零。

6.4 局部镜面反射的简单模型

这里我们只考虑不透明表面，在稍后的小节里再讨论透明表面情形。对于镜面反射，入射光线与其反射光线的关系是，入射角等于反射角。 139

在图6-5中，J是入射光线，R是反射光线。对于完全镜面反射，一定有入射角（α）等于反射角（β），而且反射光线、入射光线和表面法线在同一个平面上。因此只有当观察者沿着矢量R的方向才能看见这个特别的入射光线的反射光。Bui-Tong Phong于1975年基于这一点给出了一个近似模型，这就是著名的Phong光照模型。它引进了一种特别类型的反射，不是理想的镜面反射，称为平滑反射。反射光是从它的原点出发的一个光束，它依赖于一个发光参数，我们将在下面看到。效果如图6-6所示。

在图6-7中，L和E是从表面上一点到光源和到视点的矢量。H矢量是L和E的等分矢量。那么如果k_s为表面的镜面反射系数，h是H的规范化矢量，光照的镜面反射成分如下：

$$I_i \cdot k_s \cdot (h \cdot n)^m \tag{6-22}$$

这里m为发光参数，是一个正数。注意：

$$h = \frac{e + l}{|e + l|} \tag{6-23}$$

这里e和l分别是E和L的规范化表示。

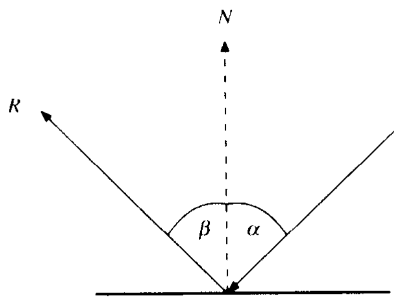


图6-5 完全镜面反射。J是入射光线，
R是反射光线。一定有$\alpha=\beta$

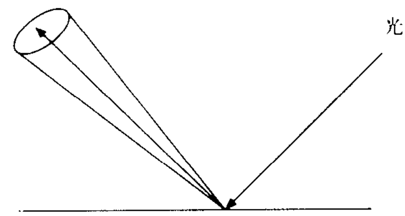
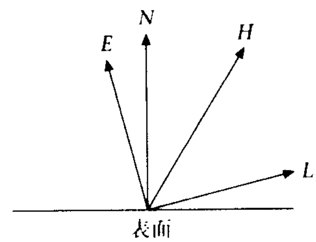


图6-6 平滑反射



L是指向光源的方向，E是指向视点的方向，H是E和L的等分方向

图6-7 镜面反射(Phone 模型)

140 式（6-22）的思想是，当入射角与反射角相等时，H和N将合而为一，所以有$h \cdot n=1$，这是可能的最大值。值m是一个常数，根据控制“发光”程度的经验确定。m会有许多值，这会产生更高阶的高光区，这是因为H和N之间角度的很小变化都会引起式（6-22）取值的较大改变。

真实表面一般既有漫反射又有镜面反射。因此为每个RGB主色，式（6-19）和式（6-22）被合在一起形成一个全面描述表面反射特性的方程，如式（6-24）所示：

$$I_r = k_a I_a + k_d I_i (n \cdot l) + I_i \cdot k_s \cdot (h \cdot n)^m \tag{6-24}$$

这里有多个光源，漫反射和镜面反射单元向先前一样被加在一起：

$$I_r = k_a I_a + \sum_{j=1}^{M} I_{i,j} (k_d (n \cdot l_j) + k_s (h_j \cdot n)^m) \tag{6-25}$$

可以对方程作进一步调整，将光能的衰减与距离的关系考虑进去——即光能与光源的距离平方成反比（式（3-11））。虽然式（6-24）中所使用的量与物理测量没有什么关系，但是这种衰减有时仍然适用。根据经验我们发现，在式（6-25）所给出的模型中严格使用这个定律并不能产生好的效果。相反，对每个代数和的成分项除以距离与一个常数的和，该常数可根据经验调整直到取得满意的结果为止。

实际上，式（6-25）的应用不是一个像素一个像素地进行的，而是采取插值的模式，并结合对像素颜色的计算以及对隐藏面删除的深度计算。这些将在第 13 章中讨论。

式（6-25）是一个经验模型，它所给出是一个可以接受的、在计算耗费和真实感之间折衷的结果。我们要再一次强调，它不是一个真实的物理模型。

6.5 用光线投射渲染局部光照模型

让我们试着把局部光照模型与前面的光亮度方程联系起来，看看能达到什么样的近似值。
141 在先前一章中，近似值是非常简单的——把所有对象都当作发射器，并令所有的 BRDF为零。这里我们可以考虑得更复杂一点。首先，注意到我们已经在场景中引进了一个新的对象类型

——点光源。在这个模型中这是惟一的光发射器，没有其他对象是发光实体，它们只反射光。在这种假设下，对于表面上的所有点 p 以及任意方向有$L_e(P, \omega)=0$。此时光亮度方程变成：

$$L(p,\omega)=\int_{\Omega} f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \,\mathrm{d}\omega_i \tag{6-26}$$

为了简化记号，我们将假设只有一个点光源。考虑图6-8 中所示的情形。视点在p点处，从某个特别的场景像素到视点的光线方向为ω。我们需要计算出$L(p, \omega)$。现在p是在自由空间中，它不是某个表面上的点，我们知道光亮度在光路上不发生改变。因此可以沿着光线回溯，直到它与第一个表面相交（如果有的话），设交点为p'。那么有：

$$L(p,\omega)=L(p',\omega) \tag{6-27}$$

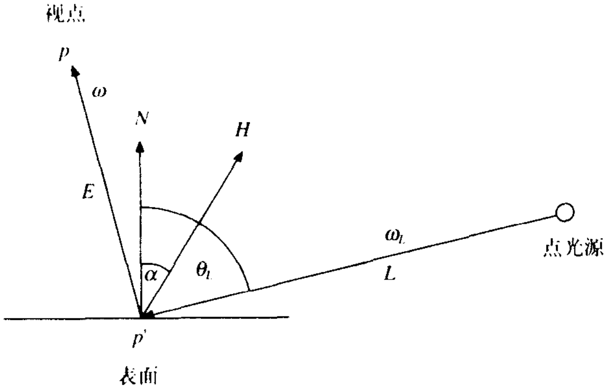


图6-8 局部光照模型

从式（6-26）中我们知道 $L(p', \omega)$可以通过在 p'处的一个光照半球上积分求得。实际上，只有一个方向上这积分是非零的：从点光源出发的光线方向(ω_L)。（其他的方向都没有贡献，因为这里只有从点光源到表面的局部光照，没有对象间的反射。）而且，在这个方向上有$L(p', \omega_L)=L_i$，也就是说，它等于来自光源所发射光的入射光亮度。

现在我们把这些结合起来，会得到：

$$\begin{aligned}L(p,\omega)&=L(p',\omega)\\&=f(p',\omega_L,\omega)L_i\cos\theta_L\end{aligned} \tag{6-28}$$

在这个结果中，BRDF 应该是什么？对表面上的任意点 p'，解应该写成：

$$f(p',\omega_i,\omega)=\delta(\omega_L-\omega_i)\left(k_d+k_s\cdot\frac{(\cos\alpha)^m}{\cos\theta_L}\right) \tag{6-29}$$ 142

这里α依赖于到视点的方向ω（事实上它是一条主光线）。除以$\cos\theta_L$是为了除去由于漫反射所造成的扩大，该项只处理镜面单元。由式（6-26），我们得到：

$$\begin{aligned}L(p,\omega)&=L(p',\omega)\\&=L_i(k_d(n\cdot l)+k_s\cdot(h\cdot n)^m)\end{aligned} \tag{6-30}$$

因此我们这里获得了几个自由：像以前那样规范化“强度”，并添加“环境光照”来获得更好效果，而且重新回到式（6-24）。

在实际中这意味着什么?它实际上是提供了对前面一章中所介绍算法的下个阶段的描述。在那里我们知道对球体的渲染所看到的却是平坦的盘子，因为每个球体是预先指定了一个颜色。现在我们可以将它处理得更复杂些。如先前一样，我们从视点出发沿着主光线回溯到场景，直到碰上了一个对象（如果发生的话），这实际上是实现式（6-27）。如果光线不与任何对象相交，那么我们将它的颜色设定为某个预先定义的背景色，例如(0，0，0)，即黑色。否则获得交点p'，我们知道点光源的位置和光强度，因而我们可以进行计算。式（6-24）对于红色、绿色和蓝色的强度分别计算，然后将这个颜色赋给相应的显示像素。

该算法有个非常重要的方面需要考虑。可能出现的情况是，相交对象表面上的点p'对于点光源可能是不可见的。有两个原因可能产生这样的情况：可能由于自身遮挡，或者是另外的一个对象遮挡了它。如何能发现是否是这种情况呢？回答是简单的——从p'沿着光线的方向跟踪光线（沿着方向$-L$，如图6-8所示）。如果这条光线在点p'和点光源之间与另外的一个对象（或者是同一个对象）相交，那么p'在阴影中，此时只有环境项是非零的。

当然，如果有不止一个点光源的话，那么我们就要使用式（6-25），方法不变。计算对每个光源重复一遍，对结果求和并与环境光照效果相加。

彩图6-9给出了只有漫反射的球体世界渲染结果。对象现在已经有了深度，不再看起来像盘子一样了。然而，它们全都看起来好像是由石灰材料做成的。彩图6-10给出了一个相同的场景，其渲染使用了Phong模型的镜面反射。现在球体看起来好像是台球似的，或者像用塑料做成的。它较之前一章的平面明暗处理的确是有进步的，但是仍然没有考虑到任何全局的对象间交叉反射的效果。我们将在下个小节中继续讨论这一点。

6.6　对递归光线跟踪的介绍

光线跟踪具体表达了光线如何在环境中传递这样一个更现实模型，这里所有的表面都是理想的镜面反射器，光发射器都是点光源（或是线光源）。该方法是由Turner Whitted于1980年引入到计算机图形学中的。它考虑到了这样的一个事实：即在表面上的一条入射光线除对
143 表面上那一点直接光照外，还进一步产生两条光线——一条反射光线和一条折射光线，这些光线将会对其他的表面光照做出贡献。同样，从其他表面来的反射也会照射到这个表面。从这个意义上讲，它是全局光照模型，而非局部光照模型：在某一特殊点上的光照不再是独自依赖于点光源和表面之间的交互，而是包括了每个表面对其他表面光照带来的效果，还要考虑到遮挡问题。在光线跟踪方面已经发表了大量的研究成果，Glassner（1989）的著作是这方面的一个很好的入门教材。

光线跟踪的基本原理是当一条光线射到一个对象的时候，它可能进一步反射出去。如果对象是透明的，那么光线将会在对象内部传导。这些反射和传导光线可能进一步撞击到其他对象，并衍生出更多反射和传导光线。暂时我们只考虑不透明表面，因为这已经足够说明光线跟踪的基本思想了。考虑图6-11，有一条光线标记为E，如前所示，从点p处出发方向为ω，本图说

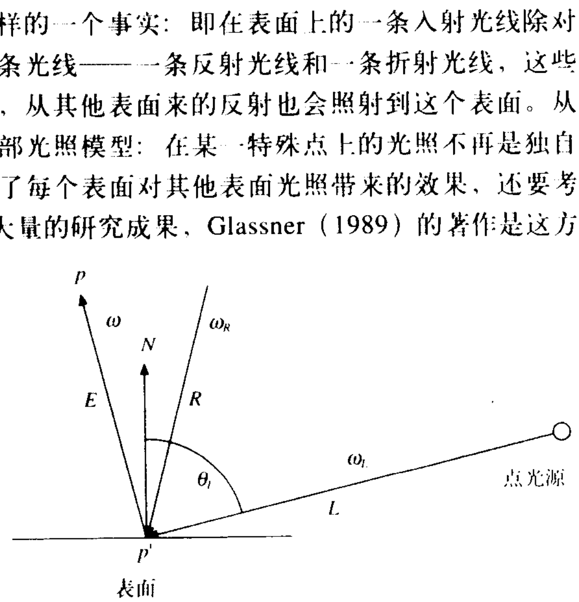


图6-11　来自反射方向的光线的贡献

明对这样一条光线的贡献情况。这样一条光线可以被当作一条主光线，p点为视点，今后没有必要再作类似的说明。

我们需要求出$L(p, \omega)$，当然此时仍然有$L(p, \omega)=L(p', \omega)$。然而，现在对于这样一条方向为$\omega$的光线有两个可能的贡献来源：一是来自点光源的方向为ω_l的光线，二是来自另外一个对象的方向为ω_R的光线，这条光线的表面反射光线正好与光线E重合。因此$L(p', \omega)$既依赖于局部点光源，如前所述，而且还依赖于$L(p', \omega_R)$。因此式（6-26）中的积分只在ω_i的两个取值处是非零的：在$\omega_i=\omega_l$时，因为此时它传递的是局部光照模型；在$\omega_i=\omega_R$时，因为它传递的是来自光线R的光（如果存在的话）。我们将BRDF写成两个成分的和的形式：

$$
\begin{aligned}
f(p',\omega_i,\omega) &= \delta((\omega_l-\omega_i)\cdot Local)+\delta(\omega_R-\omega_i)\cdot\frac{k_s}{\cos\theta_R} \\
&= \delta(\omega_l-\omega_i)\left(k_d+k_s\cdot\frac{(\cos\alpha)^m}{\cos\theta_L}\right)+\delta(\omega_R-\omega_i)\cdot\frac{k_s}{\cos\theta_R}
\end{aligned}
\tag{6-31}
$$
144

第一项只是局部光照模型的贡献。第二项允许入射光线反射的比例，该比例取决于这个表面的镜面反射系数。

通过积分运算我们得到：

$$
\begin{aligned}
L(p,\omega) &= L(p',\omega) \\
&= L_i(k_d(n\cdot l)+k_s\cdot(h\cdot n)^m)+k_s L(p',\omega_R)
\end{aligned}
\tag{6-32}
$$

又由于第一项只是局部模型，而第二项代数和来自入射光线R的影响，所以我们得到一个递归光亮度方程，这里$L(p', \omega)$依赖于$L(p', \omega_R)$。最后一项如何计算呢？可以通过重复相同的计算来达到。我们可以沿着光线R的反方向$-\omega_R$回溯，求得相交的第一表面，不妨设交点为p''，然后再求出$L(p'', \omega_R)$。每次方程的形式是相同的。这样做看起来还有一个问题，即可能会导致无休止的递归。事实上这种情况是不会发生的。一种情况是被跟踪的光线可能会不与任何对象相交，这种情况使得我们对表面赋予预先指定的颜色（比如说黑色）；否则，尽管被跟踪的光线有相交，但当这种贡献变得很小，在某个时刻可以忽略不计——在这两个情况下，递归就可以停止了。

综上所述，我们对不透明镜面反射对象有简单的递归光线跟踪算法（见算法6-1）。我们将直接使用规范化强度并引入环境项。

算法6-1 对不透明材料的递归光线跟踪

```
Color RayTrace(Point3D p, Vector3D direction, int depth)
/*ray tracing for a single light source and opaque materials*/
{
   Point3D pd;
   Vector3D R;
   bool intersection;
   Color ILocal;

   if(depth > MAX) return BLACK;

   /*intersect the ray from p in the given direction and return the
   nearest point pd along the ray. intersection is true if there is
   an intersection*/
   intersect(p,direction,&pd,&intersection);

   if(!intersection) return BACKGROUND_COLOR;
```

```
/*there was an intersection, now compute the local color at pd -
recall that this must be computed for each of R, G and B*/
ILocal = kaIa + Ip·v·(kd(n·l) + ks·(h·n)^m);
  /*where v = 1 if pd is visible to the light, else 0*/
/*continue recursion - compute reflection direction R - note that
this is -R in Figure 96*/
return ILocal + ks*Ray Trace(pd,R, depth+1);
}
```

145

现在这个函数必须对每个主光线调用一次，即对所有那些从COP点出发经过场景像素中心点的每条光线。返回值是一个颜色（RGB值），其后将用它来设定对应的屏幕像素点，如上一章所述。注意现在计算执行中所使用的方向是与光流的方向相反的——如在最后一章中所述，它是从照相机位置到场景内的。因此在算法中反射光*R*的方向是与图6-11中光线*R*的方向相反的。当然也有很多事情并未在算法中讲明——我们不仅要知道光线与表面的相交点，而且还要知道关于对象的充分信息，以便能确定它在那个交点上的法向(对于球体这种情况是很简单的）和确定它的材质属性（系数*k*)。因此实际中"求交"函数会返回一个指向对象的指针，而不是只返回一个相交点。该对象就是与光线相交的那个对象。最后我们要处理各处的RGB强度，所以对每个R、G和B必须执行局部颜色计算和误差测试。这个误差测试是用来判断所得的颜色是否很"小"，以至于不值得继续进行递归。

现在我们假定在场景中的对象都是不透明的。很明显，我们应该引进另一个项到方程中来，它允许任何光线穿透对象，如果对象是透明的，这种光线也将对表面光照有贡献。这里我们不想把这种情形引入到光亮度方程中来（我们准备把它留给读者作为练习），因为它对于我们理解所用的近似值的性质没有什么帮助。然而，在下个小节中我们将更详细地讨论光线跟踪，包含一些其他的问题。同时彩图6-12给出了球体场景下这一技术的一个例子。

6.7　包括透明对象的递归光线跟踪

我们已经看到从投影中心到场景的反向光线跟踪，而且看到了它是如何自然地随着递归光亮度方程的展开进行的。它的一个特别好处是保证了我们只跟踪最后到达视点的光线。一个特别像素上的颜色对应于观察者可见表面上的一个点的颜色，它的确定是通过对从最初的光线所衍生的所有光线的效果求和得到的，这个最初的光线就是从视点到像素的主光线。

在图6-13中我们跟踪来自视点经过一个特别像素的一条光线，它与对象A相交。从对象A表面上的交点我们跟踪两条光线到光源（这种光线我们通常称为*阴影感知器*)。注意阴影感知器s_1在到达光源L1的路径上击中对象C。因此，对于L1来说，对象A上的这个点是位于阴影中。
146 然而，它却受到来自光源L2的照射。从对象A开始，衍射出两条光线，一条是反射光(*r*)，另一条是传导光线(*t*)，因为这个表面是透明表面。考虑光线r_1，它击中对象B，对象B可看作阴影感知器，并进而又有衍生的反射光线和传导光线生成。

这个过程一直继续下去，直到光线离开环境为止（如t_2没有击中任何对象），或直到进一步的贡献对最后的颜色没有太"大"的价值。在每个后续阶段，只有一部分所接收的光能会伴随着衍生光线继续向前传递，因此实际最后光线将会逐渐衰减，对最后像素的颜色没有什么贡献。这个光线跟踪过程可以用算法6-2中的简单递归公式来描述。

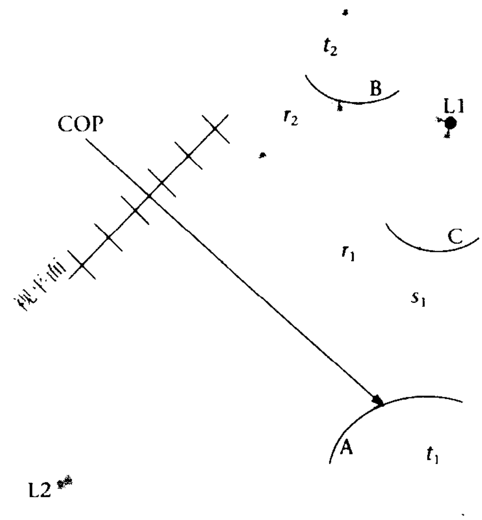


图6-13 光线跟踪

算法6-2 不透明和透明表面的递归光线跟踪

```
Color RayTrace(Point3D p, Vector3D direction, int depth)
{
  Point3D pd;
  Boolean intersection;

  if(depth > MAX) return BLACK /*(0,0,0)*/;
  else {
    /*intersect the ray from orgin in given direction
    with the scene to find the closest point of intersection
    pd. intersection = true if there is such an intersection
    */                                                          147
    intersect(p,direction,&pd,&intersection);

    if(!intersection)return BACKGROUND_COLOUR;
    else {/*l*v_i = 1 if pd is visible to the ith light, else 0*/

      Ilocal = I = k_a I_a + Σ_i I_pi · v_i((n·l_i)k_d + (h_i·n)^m k_s)

      R = reflection direction;
      Ir = RayTrace(pd,R,depth+1);
      T = transmission direction;
      It = RayTrace(pd,T,depth+1);

      return (Ilocal + kr*Ir + kt*It )
    }
  }
}
```

递归要进行到一个预先设定的深度（MAX）。光线被射入环境，计算所有与该光线相交的候选对象。在这个候选对象集合中，交点对应的是光线的最短路径，我们要把它计算出来（即*pd*）。此时布尔变量*intersection*为真，否则它为假。如果光线没有击中任何对象，那么就要赋予表面背景颜色——这意味着它已经跑出了环境。

*Ilocal*项是基于阴影感知器的，所以代数和项是对所有能到达光源而又没有与任何不透明

对象相交的阴影感知器。k_r和k_t分别为反射和传导方向的反射系数，通常会有$k_r=k_t=k_s$。

这个“局部”光照模型包括一个环境项，它代表了另一种类型全局光照，即完全不考虑光线跟踪的全局光照——漫反射的全局效果。在下个小节中我们将分析如何计算反射光线R和传导光线T。

6.8 光线跟踪算法的一些细节

反射（R）的方向

在图6-5中我们示意了入射光线(J)和反射光线(R)。N是在表面上我们所注意的那一点处的法向。计算R有两条定律可以使用（假设所有的矢量已经是规范化了的矢量）：

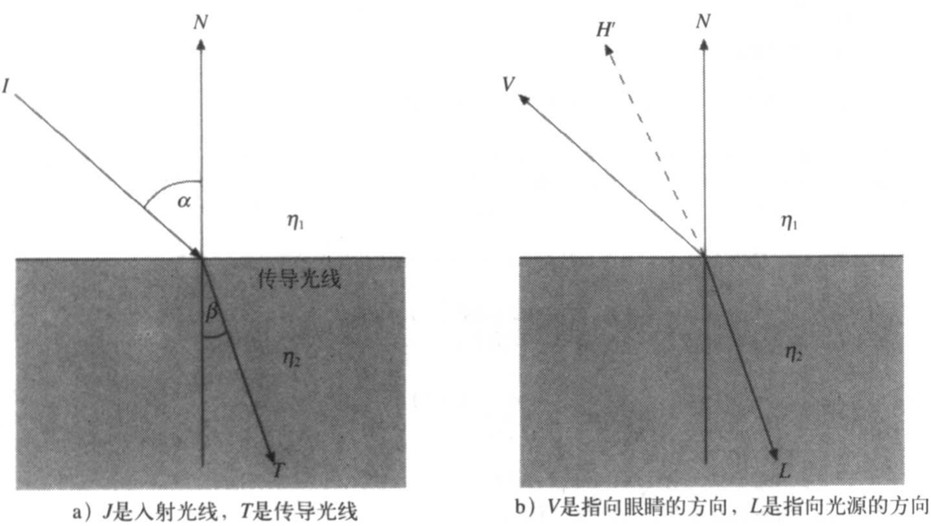
- R与J和N处于同一个平面上，所以有 $R= aJ+ bN$，a和b为某个常数。
- 角度α和β相等的。

因为 $\cos\alpha=\cos\beta$，我们有

$$-J\cdot N=N\cdot R=N\cdot(aJ+bN)=a\,(N\cdot J)+b\text{，因为}N\cdot N=1$$

又设$a=1$，因而有 $b=-2(N\cdot J)$。最后有：

148 $$R=J-2(N\cdot J)N \tag{6-33}$$



a）J是入射光线，T是传导光线　　b）V是指向眼睛的方向，L是指向光源的方向

图6-14　直接镜面传导

完全镜面传导

图6-14a)给出了当一条光线从一种媒质到另外一种媒质时所发生的情况。光线的路径发生弯曲，完全程度是由那个媒质的密度决定的，如Snell定律所描述的（Feynman et al.，1977）：

$$\frac{\sin\alpha}{\sin\beta}=\frac{\eta_2}{\eta_1}=\eta_{21} \tag{6-34}$$

这里η_2，η_1是两种媒质的折射系数。

传导光线T将会与J和N在同一个平面上，所以有$T=aJ+bN$。

使用这两条定律很容易证明（Glassner，1989）：

$$T=\eta_{12}J+N(\eta_{12}\cdot\cos\alpha-\sqrt{1+\eta_{12}^2\cdot(\cos^2\alpha-1)}) \tag{6-35}$$

现在可能出现的情况是位于根号下的值可能为负。此时被称为完全内部反射——它发生在当光线从一个密度相对较高的媒质射向密度相对较低的媒质，而且入射角度大于40度的时候。这种情况光线只有反射（这种效果发生的例子如在一个游泳池中水下的情形。当游泳者从某个角度看向水面时他将会看见来自池底部的反射）。

直接镜面传导

在光线跟踪程序的*Ilocal*项中，$(h_i\cdot n)^m k_s$代表了镜面反射成分。然而，这是假设光源位于对象的前方。另外一种可能性是对象为透明的，光线从它后面射出。 149

此时要计算矢量H'，使用Snell定律：

$$H'=\frac{V-\frac{\eta_2}{\eta_1}L}{\frac{\eta_2}{\eta_1}-1} \tag{6-36}$$

H'与先前一样是规范化了的，$k_t(h'\cdot n)^m$用来代替镜面反射项。k是镜面传导系数。这在图6-14b中说明。

相交计算

在简单光线跟踪计算中有超过90%的计算量来自于相交计算（Whitted，1980）。在先前一章中我们考虑了与一个球体的相交问题。在第8章中我们将求解与平面多边形的相交问题。

有很多关于光线跟踪的研究集中在如何减少相交计算的影响上，这些研究主要是通过减少每条光线的代价或者是减少光线的总数，或两者兼而有之。有关这个方面将在第16章中详细讨论。然而，一个明显的方法是将每个对象用一个包含整个对象的最小球体包围住，如果光线与该球体相交，那么我们就去测试光线与真实对象是否真的相交，否则的话，光线将位于球体的外面，它就根本不可能与对象相交。这是包围体方法的一个实际应用。

6.9 OpenGL中的光照

OpenGL提供了对上面所描述的光照模型的一个简单表示。它支持环境光、漫反射和镜面反射材质属性，以及支持多个光源。在这一小节中我们将要考虑关于材质描述和点光源的多个例子。在OpenGL中系统引进“状态”的概念，借此来给出当前材质的属性，在状态中的材质属性将保持不变，直到它们被显式地改变。在构造材质的数据结构方面，所采取的策略是对对象表面多面体的每一个面赋予一种“材质”，举例来说，构成比较复杂对象的多边形。然而，虽然这种做法比较简单，但它隐含着一个极大的效率问题，因为这种一个多边形一个多边形地处理将极大地降低OpenGL的渲染速度。所以，我们采用基于整个对象的材质属性策略，但是允许对单个多边形材质属性进行重新赋值。如果对象没有选择材质，那么就赋予它一个默认值。 150

我们首先构造一个`Material`数据结构，然后说明如何在OpenGL中使用。

```
#define NOMATERIAL ((Material *)0)
typedef enum{
  FlatShading, SmoothShading
} ShadingModel;

typedef enum{
  Opaque, Transparent
} Opacity;

typedef struct{
  GLenum model;              /*GL_FLAT or GL_SMOOTH*/
  GLfloat ambient[4];        /*rgb alpha*/
  GLfloat diffuse[4];        /*rgb alpha*/
  GLfloat specular[4];       /*rgb alpha*/
  GLfloat shininess[1];      /*shininess for specular*/
  Opacity opacity;           /*transparent means glEnable(GL_BLEND)*/
} Material, *MaterialPtr;
```

Material结构表示了上述模型的每个成分，包含了环境反射、漫反射和镜面反射系数。“发光参数”是式（6-23）中的m，它表示为一个数组，有一个元素方便用于与 OpenGL 描述的比较。这个表示也允许透明性设置和两种不同类型模型：平滑明暗处理和平淡明暗处理。我们将不在这一章中考虑这些问题。在 OpenGL 中，像素实际上是 RGBA 值，这里“A”代表“alpha”。对它的解释将在第23章中说明。

```
void setDefaultMaterial(void)
/*sets a material for use when both object and face have NULL
materials*/
{

  GLfloat ambient[] = {0.2,0.2,0.2,1.0};
  GLfloat diffuse[] = {1.0,0.0,0.0,1.0};
  GLfloat specular[] = {1.0,1.0,1.0,1.0};
  GLfloat shininess[] = {20.0};

  glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,diffuse);
  glMaterialfv(GL_FRONT,GL_SPECULAR,specular);
  glMaterialfv(GL_FRONT,GL_SHININESS,shininess);
  glShadeModel(GL_FLAT);
}
```

151 在setDefaultMaterial函数中，我们使用OpenGL的glMaterial函数设定反射的各种系数。像先前一样，注意“f”代表浮点数，而“v”代表矢量值。举例来说，环境反射系数由四个浮点数矢量表示。前三个是红色、绿色和蓝色成分，最后一个称为“alpha”成分，可以用来描述透明性（见第23.4节）。

```
void setMaterial(Material *material)
/*sets the current material in OpenGL*/
{
  glMaterialfv(GL_FRONT,GL_AMBIENT,material->ambient);
  glMaterialfv(GL_FRONT,GL_DIFFUSE,material->diffuse);
  glMaterialfv(GL_FRONT,GL_SPECULAR,material->specular);
  glMaterialfv(GL_FRONT,GL_SHININESS,material->shininess);
  glShadeModel(material->model);
  if(material->opacity==Transparent) glEnable(GL_BLEND);
  else glDisable(GL_BLEND);
}
```

在setMaterial函数中，通过所提供的Material参数将OpenGL设置到状态中。

```
static void initialize(void)
{

   GLfloat light_ambient[] = {0.2,0.2,0.2,1.0};
   GLfloat light_diffuse[] = {1.0,1.0,1.0,1.0};
   GLfloat light_specular[] = {1.0,1.0,1.0,1.0};

   GLfloat light_position[] = {0.5,1.0,1.0,0.0};

   /*GL_FLAT or GL_SMOOTH*/
   glShadeModel (GL_SMOOTH);

   /*set the background (clear) Color to white*/
   glClearColor(1.0,1.0,1.0,0.0);

   /*enable normalization*/
   glEnable(GL_NORMALIZE);
   glEnable(GL_DEPTH_TEST);

   /*set the depth buffer for clearing*/
   glClearDepth(1.0);

   /*enable lighting*/
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0,GL_AMBIENT,light_ambient);
   glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
   glLightfv(GL_LIGHT0,GL_SPECULAR,light_specular);
   glLightfv(GL_LIGHT0,GL_POSITION,light_position);

   /*set up camera and scene here...*/
}
```

152

函数initialize是一个有关设定OpenGL各方面初值的代码片段，包括光照。光（I_a、I_{pd}，I_{ps}）的环境反射、漫反射和镜面反射成分在这里定义（注意 OpenGL允许镜面和漫反射成分是不同的），通过使用glLightfv设置，它也能用来设定位置。注意光照一定要被激活方可使用（glEnable)。我们将在稍后讨论各种类型的明暗处理模型（平滑的或平淡的）。

6.10 VRML97中的光照

在VRML97中表面材料模型所使用的参数几乎可以直接映射到OpenGL上。表面材质属性的描述是通过一个Material节点来完成的，它本身是Appearance节点中的一个域。这个外观节点控制几何对象的全部外观，它通过定义材质、纹理和纹理变换属性。材质节点的六个域分别是diffuseColor、specularColor、emissiveColor、ambientIntensity、shininess和transparency。前三个定义为 RGB三元组，其他三个都是位于0和1之间的浮点值。

两个基本成分是diffuseColor和specularColor，它们的功能从命名即可看出。注意这里没有“ambientColor”域，但是有一个代替它的变量，这就是ambientIntensity，它是单一的浮点值，定义了这个材料将会反射场景中总的环境光的多大比例。环境颜色可以由ambientIntensity*diffuseColor确定。shininess域的作用与它在OpenGL中的相同，即式（6-22）中的m，不同之处是在VRM 中它是在0和1范围内，而在OpenGL中它的范围为0

和128之间。对于所有的颜色成分都使用一个transparency值。最后，emissiveColor域可以被用来定义独立于世界中光照的表面颜色。这在辐射度类型应用中是很有用的，因为在这种应用中全局光照解是预先计算的，并被“写”到几何上。

材质节点中的三个域有非零缺省值。diffuseColor有一个缺省的颜色为（0.8 0.8 0.8），ambientIntensity的缺省值为0.2，而shininess的缺省值为0.2。

在图6-15a中我们定义了三个球体。第一个是暗绿色，第二个是亮蓝色，而第三个是一个
153 透明的、自己发光的红色球体（见彩图6-15b）。

我们在图6-15中还定义了惟一一个DirectionalLight节点，以便说明局部明暗处理的效果。场景设计者还可以使用PointLight节点和SpotLight节点。这些内容请参见之后的“光”这一小节，在那里说明了VRML中光照的概况。如果在VRML文件没有光的定义，那么VRML浏览器就会创建一个“头灯”，即一个与视点相连的线光源，它的方向总是与视线方向一致。

```
#VRML V2.0 utf8
Group {
 children [
  DirectionalLight {
  color 1 1 1
 }

# Dull green sphere
 Shape {
  appearance Appearance {
   material Material {
    ambientIntensity 0.1
    diffuseColor 0.1 0.7 0.2
    shininess 0.0
   }
  }
  geometry Sphere {
   radius 1
  }
 }

# Shiny blue sphere
 Transform {
  translation 3 0 0
  children [
   Shape {
    appearance Appearance {
     material Material {
      ambientIntensity 0.3
      diffuseColor 0.2 0.1 0.7
      specularColor 0.7 0.8 0.6
      shininess 0.6
     }
    }
    geometry Sphere {
     radius 1
    }
   }
  ]
 }

# Transparent red sphere
 Transform {
  translation -3 0 0
  children [
   Shape {
    appearance Appearance {
     material Material {
      diffuseColor 0.0 0.0 0.0
      emissiveColor 0.7 0.2 0.1
      transparency 0.5
     }
    }
    geometry Sphere {
     radius 1
    }
   }
  ]
 }
 ]
}
```

图6-15　a）VRML 97材质示例

6.11　小结

本章介绍了计算机图形学中光照计算的基本思想及其表示，覆盖了完全漫反射和镜面反射表面的思想，并导出了点光源的光照方程。还讨论了全局光照的方法，即光线跟踪，它提
154 供了一个简单的镜面-镜面相互反射的建模方法。我们了解到局部光照模型和光线跟踪方法提供了光亮度方程的一种特别类型的解法。稍后的几章将会进一步讨论全局光照以及快速计算方法。我们还介绍和总结了在OpenGL及VRML中如何获得明暗处理模型。讨论至此还是在上一章中提出来的那个简单的照相机观察模型的上下文中进行的。我们将在下一章中扩充这一模型。

第7章 照相机的一般化

7.1 引言

在先前的两章中我们使用了一个非常简单但又受限制的观察模型:视线方向是沿着负 Z 轴的方向，视点COP被固定在正Z轴的某处，照相机总是在垂直方向上与正Y轴对齐。通常我们需要一种能从任何位置观察场景的能力，也就是需要照相机可以摆放在任何位置。现在就说明该如何做到这一点。

在3D世界坐标中的场景是用右手坐标系来描述的，这一点在前面我们已经介绍过。我们在前面所引进的简单抽象照相机假设为沿着Z轴朝向场景的，这是一个限制性很强的假设。这里我们介绍一组参数允许照相机朝向任意方位。策略是将变换应用到照相机，相应地也将对场景进行变换，将情形简化为图5-8中所示。我们将重新用图7-1说明。 155

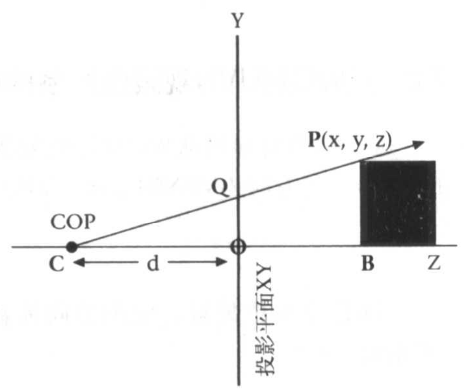


图7-1 简单的照相机模型

为了构造这样的照相机，我们定义一个新的坐标系，以照相机为中心角色。这被称为**观察坐标系**（VC），在本书中有时也叫做**眼睛坐标系**。它是一个以视图为中心的坐标系，而非WC，它的原点位置在空间中是任意的。这是一种左手坐标系（从右手坐标系到左手坐标系的转变是很简单的事情）。

有多个参数用于定义观察坐标系。它们分别是:

视图参照点（VRP）。这是WC中的一个点，它定义了新观察坐标系的原点。直观上我们可以把它看成场景中的一个相关点，或者是一个描述“照相机”（即视平面和投影中心）的点。

视平面法向（VPN）。这是WC中的一个矢量，它的方向规定了观察坐标系的正 Z 轴方向。因此这个轴是经过VRP并与VPN平行的。直观上看，它可以看成是抽象照相机所朝向的方向。新系统的Z 轴被称为 N 轴。视平面与这个矢量保持垂直。

视图上方矢量（VUV）。WC中这个矢量定义了新坐标系的正Y轴方向。Y轴是将VUV向垂直于VPN并穿过VRP点 的平面投影所得的。这个投影是新坐标系的Y轴。Y轴也称为V轴。 156

最后，再定义新坐标系的X轴。X轴是这样构造的，给定Y轴和Z轴，则X轴与它们满足左手法则（或右手法则）。X 轴通常也称为U轴。

观察坐标系的三个主要轴的名字又给了该坐标系另外一个名字——UVN系统。这些思想在图7-2中说明。我们应该清楚地认识到新观察空间是最初的世界空间的一个平移和旋转——这两个空间有一样的长度单位。场景的描述有两个不同的参考框架: 场景中的量度没有发生改变，只有场景描述所依赖的坐标发生了改变。

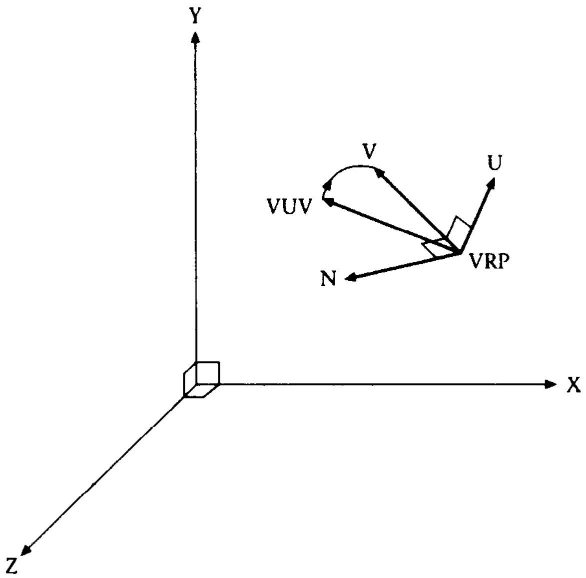


图7-2 UVN坐标

7.2 从WC到UVN观察坐标系的映射

这一小节将导出从WC空间映射到观察坐标系的变换矩阵。设n是单位向量（相对于WC的原点），其方向为VPN的方向。因此，

$$n = \frac{VPN}{|VPN|} \tag{7-1}$$

设u是个单位矢量，它的方向为新系统的U轴方向。因此，为了要形成满足左手法则的VC系统，有：

$$u = \frac{n \times VUV}{|n \times VUV|} \tag{7-2}$$

这里 × 代表矢量叉乘。最后求V轴方向的单位矢量 v，

$$v = u \times n \tag{7-3}$$

现在设M是所需的 4 × 4 矩阵，将WC空间映射为VC空间。M可以被分割成一个3 × 3的纯粹旋转部分R，以及一个平移矢量t，如下所示：

157 $$M = \begin{bmatrix} R & 0 \\ t & 1 \end{bmatrix} \tag{7-4}$$

由式（7-2）到式（7-4）所导出的矢量 u，v，n 必须用变换矩阵R旋转成VC空间的单位主矢量 i，j，k，这里 $i=$（1，0，0），$j=$（0，1，0），$k=$（0，0，1）。因此，

$$\begin{aligned} uR &= i = (1,0,0) \\ vR &= j = (0,1,0) \\ nR &= k = (0,0,1) \end{aligned} \tag{7-5}$$

写成：

$$\begin{bmatrix} u \\ v \\ n \end{bmatrix} R = I \tag{7-6}$$

这里I是 3×3单位矩阵。因为 u，v和 n是标准正交矢量（它们每个的模都为1，任意两个的点乘为0）：

$$\begin{aligned} R &= (u^T, v^T, n^T) \\ &= \begin{bmatrix} u_1 & v_1 & n_1 \\ u_2 & v_2 & n_2 \\ u_3 & v_3 & n_3 \end{bmatrix} \end{aligned} \tag{7-7}$$

为了获得平移矢量t，我们必须将VRP转换到VC系统的原点。设 VRP 由q表示：

$$(q,1)M = (0,0,0,1) \tag{7-8}$$

因此，由式（7-4）得

$$qR+t=0 \tag{7-9}$$

所以有：

$$\begin{aligned} t &= -qR \\ &= -\left(\sum_{i=1}^{3} q_i u_i, \sum_{i=1}^{3} q_i v_i, \sum_{i=1}^{3} q_i n_i \right) \end{aligned} \tag{7-10}$$

将上面各式结合起来便得到从WC到VC转换矩阵：

$$M = \begin{bmatrix} u_1 & v_1 & n_1 & 0 \\ u_2 & v_2 & n_2 & 0 \\ u_3 & v_3 & n_3 & 0 \\ -\sum q_i u_i & -\sum q_i v_i & -\sum q_i n_i & 1 \end{bmatrix} \tag{7-11}$$ 158

现在我们还要求出M矩阵的逆矩阵，将观察坐标转换回世界坐标利用逆矩阵是必要的。这是较为容易计算的。从式（7-4）中我们知道R是一个正交矩阵，它的转置矩阵就是它的逆矩阵。我们可以更简洁地将式（7-11）写成：

$$M = \begin{bmatrix} R & 0 \\ -qR & 1 \end{bmatrix} \tag{7-12}$$

从这可以很容易看到：

$$M^{-1} = \begin{bmatrix} R^T & 0 \\ q & 1 \end{bmatrix} \tag{7-13}$$

这可以通过两矩阵相乘得到单位矩阵来证明：$M \cdot M^{-1}=I$。

7.3 在光线跟踪中使用一般照相机

现在在式（7-12）中定义的矩阵M会将任何WC点 $p=(x, y, z)$映射为相同的点（它的真实位置不发生改变），但是重新描述为在 UVN 观察坐标系中。$(p, 1)M=(r, 1)$，这里r是VC中所表示的点。

投影中心（COP）是在VC系统中表示的。同样地，视平面以及每个场景像素的中心点都是相对于VC系统给出的。所以实现一个任意照相机的方法首先是将所有场景（WC）对象进行变换、将它们重新表示为在VC中，然后再执行第6章中的光线跟踪。

对于一个完全由球体构成的场景，这是一个代价很小的操作。我们知道从WC到VC的变换不在尺寸上发生任何改变，它只是有平移和方位的变化。所以球体的半径不受影响，只有中心位置要做相应的改变。因此，将*M*应用到每个球体的中心，然后完全同样地执行先前的光线跟踪计算。这是任意照相机渲染场景问题的一个非常简单而精致的解决方法。

还有另外一种处理方法。如前所述，COP和每个场景像素的位置都是VC中的点。如此，使用逆变换矩阵，如在式（7-13）中所定义的，应用到每条光线将会得到完全相同的效果。所以在这种情形中，对象没有被转换，而是将每条光线从VC 转换到了WC。

159 哪一个方法会是最有效率的呢？光线跟踪是一件很慢的事情，所以这是一个很重要的问题。光线的数量有可能比对象数量多好多个数量级。对于一个十分复杂的场景，假设有500 000个对象的场景（无论如何光线跟踪不太可能应用在这样的场景中）。但是如果显示分辨率（场景像素的数目）为1 000 × 1 000，那么这已经是1 000 000条主光线，更不用说所有来自反射和传输结果所衍生出的第二代光线了。很显然将对象变换到 VC 中，然后在 VC 空间中执行光线跟踪是更有效率的方式。

7.4　VRML97例子

观察VRML场景的照相机性质是用Viewpoint节点来定义的。该节点包含了平移、方向和一个视图域。如果VRML文件是通过一个插件在网页上看的，则视平面窗口的尺寸将在HTML文件中配置，若在VRML浏览器中看就由浏览器本身来配置。外观比例将自动设定，以便视图不会发生变形。

```
Viewpoint {
  eventIn  SFBool  set_bind
  exposedField SFFloat  fieldOfView  0.785398  #(0,Π)
  exposedField SFBool  jump  TRUE
  exposedField SFRotation orientation  0  0  1  0  #[-1,  1],(-∞,∞)
  exposedField SFVec3f  position  0  0  10  #(-∞,∞)
  field  SFString  description  ""
  eventOut  SFTime  bindTime
  eventOut  SFBool  isBound
}
```

标准照相机参数可以推断如下。平移和定向域与当前变换相关联，当前变换来自于在场景图中的Viewpoint位置。所以VRP 是这个组合变换的局部坐次原点，VUV为+Y，VPN为-Z。注意缺省的照相机位置为0 0 10，方向为0 0 1 0，且没有旋转。这样VPN在世界坐标系中为（0 0 -1），VUV为（0 1 0）。

域description定义了照相机的名字。这个名字通常放在视点列表中，以便用户能通过选择视点名字跳到这个视点上。

域bindTime、isBound、jump和set_bind允许在场景的多个视点之间切换。对如何实现这些的讨论超出了本书的范围，我们将在后面章节中对如何用VRML编程作一般性的介绍。

7.5 小结

在这一章中我们考虑的问题是：如何定义一个任意照相机的位置和方向。我们定义了观察坐标系作为照相机的相关坐标框架。这是通过三个参数表示的，分别为：新坐标框架的原点（VRP）、视图方向（VPN）以及在图像中"垂直"的轴线（VUV）。这三个参数允许我们构造WC和VC之间的变换矩阵。由此光线跟踪可以在新的坐标系中完成，所有的对象将被转换到新坐标系中。

这使得光线跟踪系统比先前所描述的要更加强大。然而，我们仍然被限制在相对简单的场景范围内，而场景的主要对象为球体。

我们说明了该如何用VRML定义一个视图，但是没有继续说明如何使用OpenGL来做类似的工作，这需要我们对后面章节中所介绍的有关观察过程有更深入的理解才行。 160~161

在下一章中我们将会扩大基本体素的范围，将包括多边形和多面体，并说明该如何用复合对象组成场景。

第8章 场景构造

8.1 引言

在这一章中我们讨论 3D 虚拟环境的几何表示这个基本问题。真实世界从几何角度看是非常复杂的——环顾一下你的周围，思索一下如何用几何描述这个真实环境便可想像出它的难度。所以我们必须对现实世界进行抽象，去除其复杂性和具体细节，将对象及它们之间的关系通过一组简化假设和适当的数学结构来描述。我们在本章中将要给出的假设，同时也是计算机图形学中的核心假设，即真实世界的几何外观是可以用平面多边形来充分表现的。在计算机图形学中有多种表示形式，包括曲面表面和不规则面片表示，人们通常采用不规则面片来表示如岩石和山脉等一些形状的自然对象，但是多边形表示是计算机图形学实践中的一种标准而通用的表示方法。举例来说，绝大多数图形硬件加速器只能处理多边形。

162 此时我们必须搞清楚表示和渲染之间的本质区别。所谓表示是描述对象或场景的方法，它是独立于对场景的任何具体渲染手段的。场景的表示通常在外部它表现为一个文件，包含了几何和其他数据，而在程序运行期间，它又表现为在内存中的数据结构形式。渲染是假设有一个视点和方向，我们如何显示出所看到场景的问题（设有一个朝向特定方向的照相机）。最后，无论用什么表示形式，通常这些表示都要映射为多边形形式，以便进行渲染（原因前面已经提到过的，那就是绝大多数的图形硬件只能处理多边形）。因此，设想某个场景是用一组曲面来描述的，当渲染这些场景的时候，表面通常被很多小多边形所代替。在这一章中我们只关注表示问题，不考虑渲染。

在下个小节中考虑如下几个问题：首先，我们所考虑的所有多边形皆为平面多边形——它们的顶点全部位于同一个平面上。掌握这个平面的方程对于很多操作（诸如可见性判定和照明）都是至关重要的，我们还要回顾一下这方面的数学内容。其次，我们要介绍多面体的概念，一种表现对象的特殊类型，通过连接多个多边形构成一个对象的外观描述。我们介绍一个简单而先进的数据结构来表示多面体。假设能描述单个对象，我们考虑一个场景——对象的集合是如何通过层次结构组织在一起的。这需要理解对对象的变换，以及必须的数学知识。最后，我们给出一些描述场景的真实系统的例子，尤其是虚拟现实建模语言VRML。

8.2 多边形和平面

平面方程

一个多边形由一个点或顶点的序列定义：

$$[p_0, p_1, \cdots, \cdots p_{n-1}], p_n \equiv p_0 \qquad (8\text{-}1)$$

163 且$p_i=(x_i, y_i, z_i)$。每个p_{i-1}到p_i，$i=1, \cdots, n$ 是一条边。

假设点是共平面的，也就是说，它们全部位于相同的平面上。任何三个不同点总是在同一个平面上。然而，第四个点却未必存在于前三个点所确定的那个平面上。因此，要求多边

形的所有点位于相同的平面上是一个非常强的限制。这是为什么我们在计算机图形学实践中偏爱三角形的缘故，因为显然三角形的三个顶点总是共平面的。凸多边形是最简单的一类多边形，其保持了三角形的某些简单性：每个内角都小于180度。三角形显然是一个凸多边形。在计算机图形学中对凸多边形的强烈偏爱还由于它非常容易被分解成一组三角形。计算机图形学常常将多边形也称作**面**——我们通常将一个复杂的对象看成是由很多块"面片"所包裹的或所构成的，每个这样的面片都是一个平面多边形。在这一章中我们将不加以区别地使用"多边形"和"面"两个概念。

平面方程的形式是

$$ax+by+cz=d \tag{8-2}$$

这里x、y和z是坐标，a、b、c和d是已知的系数。平面上的任何点（x，y，z）必须满足该方程，反之，满足该方程的任何点必定存在于这个平面上。我们现在将说明这几个系数的含义，我们通过任意三个不同点来构造平面的方程，在这个过程中解释清楚系数的意义。

假设平面就是本页书的表面。考虑在这个平面上的三个点p_0、p_1、p_2，如图8-1所示，任何其他一个也在该平面上的点 $p=(x, y, z)$。点p可以被认为是一个自由变量。

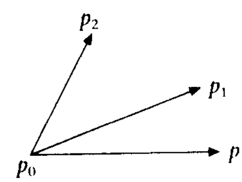


图8-1　三点定义一个平面

叉乘$(p_1-p_0)\times(p_2-p_0)$定义了一个矢量（n），它与该平面保持垂直，指向读者。这样的一个矢量称为平面的**法向量**。显然一个平面有两个方向相反的法向量。现在考虑矢量$p-p_0$。因为这个矢量与矢量n垂直，所以其内积必为零，$n\cdot(p-p_0)=0$。

既然p是平面上的任意点，我们有关于多边形的平面方程如下：

$$[(p_1-p_0)\times(p_2-p_0)]\cdot(p-p_0)=0 \tag{8-3}$$

或

$$n\cdot p=n\cdot p_0$$

平面方程的一般形式是式（8-2）。如果设 $n=(n_1, n_2, n_3)$，那么显然有$a=n_1$，$b=n_2$，$c=n_3$，它说明法向量是(a, b, c)。

而且，

$$d=n\cdot p_0=n_1\cdot x_0+n_2\cdot y_0+n_3\cdot z_0 \tag{8-4}$$ 164

注意到对点的标记顺序是十分重要的。如果我们使用式（8-3），但是交换点p_1和p_2的空间位置，那么所得的方程当然仍然是个平面方程，但是其法向将是反向的，即远离读者的方向。

这一点是非常重要的——因为一个平面显然有两侧，这两侧在数学描述上没有任何区别。然而，当我们要使用平面多边形来表示对象表面的时候，就需要知道多边形（或平面）的哪一面是表面的"外侧"，哪一面是"里侧"。式（8-3）给了我们一个准则来判断一个多边形的"外侧"面：当我们正面对着多边形的外侧时，多边形的顶点按反时针方向排列。这样当我们任意选择三个相继顶点用于式（8-3）中时，法向将朝向我们。

有关平面方程的一个非常有用的事实是，它允许我们确定平面的其他顶点在空间中的关系。根据式（8-2），设：

$$l(x,y,z)=ax+by+cz-d \tag{8-5}$$

那么显然平面方程也可以写成 $l(x,\ y,\ z)=0$。现在平面将整个 3D 空间分成了三个不相交区域：在该平面一侧所有点所构成的半空间、在平面上的所有点以及平面另一侧的所有点所构成的半空间（平面是无限的）。

包含法向量$(a、b、c)$的半空间称为正半空间，而另一半空间称为负半空间。之所以这样命名，是因为正半空间所包含的所有点满足$l(x,\ y,\ z)>0$，而负半空间中所有点都有$l(x,\ y,\ z)<0$。

现在假设$(X、Y、Z)$是任意点。那么下列陈述为真：

如果$l(X、Y、Z)>0$，那么$(X、Y、Z)$位于正半空间（平面法向量$(a、b、c)$所指一侧的点与该点在平面的同一侧）。

如果$l(X、Y、Z)<0$，那么$(X、Y、Z)$位于负半空间中。显然，如果$l(X、Y、Z)=0$，那么$(X、Y、Z)$正好位于该平面上。

记得对平面方程计算的构造是能保证矢量$(a，b，c)$指向平面的外侧，这些事实给了我们确定一个点是位于平面“外侧”还是“内侧”的方法。这对于隐藏面删除是非常有用的。

光线与多边形相交

从前一章便知，求一条光线（一条直线）和一个多边形的交点（如果存在的话）是十分重要的。这个过程有两个步骤，第一步是非常容易的，而第二步相对来讲较为复杂。第一步
165 是求出光线和多边形所在平面的相交点。只有一种情况会找不到这样的交点，即光线与平面平行的情况。

假设平面方程是式（8-2），而且光线源点是 $q_0=(u_0,\ v_0,\ w_0)$，方向矢量为$dq=(du,\ dv,\ dw)$，那么光线的参数化方程是：

$$q(t)=q_0+t\cdot \mathrm{d}q,\ t\ge 0 \tag{8-6}$$

这里光线与平面相交，因而：

$$a(u_0+t\,\mathrm{d}u)+b(v_0+t\,\mathrm{d}v)+c(w_0+t\,\mathrm{d}w)=d \tag{8-7}$$

合并同类项求得t为：

$$t=\frac{d-au_0-bv_0-cw_0}{a\,\mathrm{d}u+b\,\mathrm{d}v+c\,\mathrm{d}w} \tag{8-8}$$

当光线与平面平行的时候分母将会为零，此时没有相交。否则，将t代入式（8-6），我们求得点$p=(x、y、z)$，这是光线与多边形平面的交点。（我们将在第 10 章中继续讨论另一种情况下的直线与平面的相交问题。）

现在假定我们已知点 p，虽然它是在多边形的平面上，但它可能不在多边形的内部。因此现在的问题变成已知3D空间中的一个点以及在3D空间中的一个多边形，如何确定是否点在多边形的内部。通常在3D空间中这不是一个容易的问题。我们要将这个问题转移到2D空间中处理，将多边形的顶点和点p投影到一个主平面（XY，XZ或 YZ）上，这样做会容易得多。投影后点与多边形之间的关系并没有改变，即原先的点若落在多边形的内部，则投影后其仍将在投影多边形的内部，反之亦然。

那么多边形应该投影到三个主平面中哪个平面上呢？需要避免的一种极端情形是所选择的平面垂直于多边形平面，因为此时投影会退化为一条直线。一个好的选择是让主平面与多边

形的平面基本平行。更准确的描述是多边形平面的法向和所选择主平面法向之间的角度应该到达最少。角度的最小化可以通过对法向点乘取最大值求得（假设这些法向量都是规范化了的）。表8-1给出了主平面相应的点乘值，对于矢量$n=(a, b, c)$，相应于XY、XZ和YZ的点乘值分别为c、b和a。因此所选择的主平面应该是对应于平面方程中系数的绝对值最大的那个。投影过程本身是简单的：如果选择的是XY平面，那么就将多边形的各顶点z坐标变为零，同时把p点的z坐标变为零；如果选择的是XZ平面，则相应地将它们的y坐标变为零；若选择的是YZ平面，则相应地将它们的x坐标变为零。换句话说，如果$|a|$为参数中的最大值，则我们的投影操作就是把x坐标降为零；如果$|b|$为最大值，则把y坐标降为零；如果$|c|$是最大值，则把z坐标降为零。

不失一般性，让我们假设所选择的主平面为XY平面，被投影的点是$p'_i=(x_i, y_i)$。我们以图8-2中所描述的情形结束这个问题的讨论。一般来讲，如果多边形可以是任何形状，即使是在二维空间上确定一个点是否位于多边形的内部，也是一件很困难的事情，我们将在第12章中更详细地讨论这一点。这里假设多边形是凸多边形。

表8-1 主平面的法向以及与$n=(a, b, c)$的点乘

主平面	平面方程	法向n_{pp}	$n_{pp}\cdot n$
XY	$z=0$	(0,0,1)	c
XZ	$y=0$	(0,1,0)	b
YZ	$x=0$	(1,0,0)	a

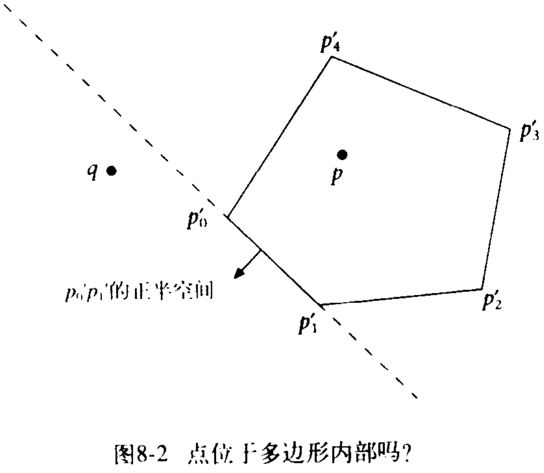


图8-2 点位于多边形内部吗？

我们假设多边形顶点按反时针方向顺序排列，如图8-2。假设$p'_i=(x_i, y_i)$，那么很容易证明多边形边的直线方程是： 166

$$\begin{aligned} e_i(x,y) &= (x-x_i)\,\mathrm{d}y_i-(y-y_i)\,\mathrm{d}x_i=0 \\ \mathrm{d}x_i &= x_{i+1}-x_i \\ \mathrm{d}y_i &= y_{i+1}-y_i \end{aligned} \tag{8-9}$$

考虑边$e_0(x, y)$，这是图8-2中虚线所示的直线。这条直线的正半空间，即与直线的法线位于同一侧的所有点是那些(x, y)，满足$e_0(x, y)>0$。相应于多边形边的每个直线方程也都是一样的。假设顶点都是反时针方向排序的，而且多边形是凸的。现在考虑一个点比如为q，它位

于多边形的外面。将会有一些$e_i(q)$值将为正（对于第0条边和第4条边），其他值将为负。然而，对于位于多边形内部的任意点，所有的值都将为负（一个点在多边形的内部，则它将位于每
167 条边的负半空间中——或在其中的一条边上）。

所以确定一个点是否落在多边形内部的算法可以按照下列规则构造：如果对于每条边，i=0、1、…、$n-1$，有$e_i(p)<0$，那么点位于多边形的内部，否则它在外部。

有关光线与多边形相交的研究稍微有点离题，我们将回到本章的主要问题上来，即基于多边形的3D场景表示。

8.3 多面体

3D场景中的对象总是由一个庞大的多边形集合来表示。一般每个多边形属于一个更大的结构——多面体，其中多边形间通过边联系在一起。简单多面体有下列各项性质：

- 每条边正好连接两个顶点，且正好为两个面的共同边界。
- 每个顶点至少为三条边的交点。
- 除了沿着两个面的公共边之外，它们不再有别的相交。

多面体的边数（E）、面数（F）和顶点数（V）总是满足欧拉公式(假设多边形没有空洞)：

$$V-E+F=2 \tag{8-10}$$

我们在这里不考虑更复杂的结构，例如有空洞的多面体。而且，假设有多个面相交于一个顶点，如果我们沿着包围这些面的一条封闭路径一圈，这条路径不应断开，而且不会与公共顶点相交。这就排除了一些特殊情况，举例来说，两圆锥体在它们的顶点处相交——这种结构在这个定义下不是一个多面体。关于多面体的数学描述请参见Coxeter（1973）。

多面体是对象的边界表示的一个例子，也就是说，边界表示将对象用描述其边界的面的几何信息来表示。对象还有其他形式的表示，例如八叉树表示、二叉空间分割树、体素构造表示等，这些将会在稍后讨论。

多面体的顶点-面数据结构

表示多面体的最为简单的数据结构是独立多边形的集合表示。这从空间上讲是效率不高的，从建模的角度看也是不实用的。举例来说，
168 很显然，每个顶点将至少被存储三次，每个边至少被存储两次。下面将要介绍的是**顶点-面数据结构**，并给出实例。

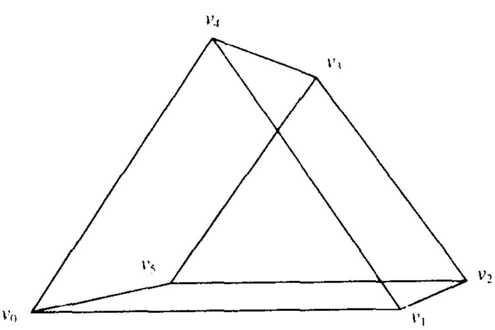


图8-3　多面体的例子

图8-3中的楔形多面体有6个顶点、9条边以及5个面($V-E+F=2$)。将这些面标记为：

$v_0v_1v_4=F_0$

$v_5v_3v_2=F_1$

$v_1v_2v_3v_4=F_2$

$v_0v_4v_3v_5=F_3$

$v_0v_5v_2v_1=F_4$

数据结构包括两个序列，一是对象中所有顶点的序列，另一个是所有面的序列，如下表

所示:

顶点	面
v_0	0、1、4
v_1	5、3、2
v_2	1、2、3、4
v_3	0、4、3、5
v_4	0、5、2、1
v_5	

这个数据结构的实现是很简单的。顶点可以用一个动态分配的3D点的数组来存储。而一个面因而可以用一串整数来表示，每个整数代表了顶点数组中的一个点。举例来说，假如顶点数组中第i个元素用$v[i]$表示，那么上面例子中的第一个面所包含的顶点是$v[0]$、$v[1]$、$v[4]$。有一点十分重要，那就是面的顶点顺序，因为它隐含了面的方向信息。顶点一定要表示为这样的顺序，当我们朝向面的正向时，顶点是反时针方向顺序排列的。这是计算平面方程的函数构造所决定的，它保证了面的法向和顶点顺序遵从右手法则（换言之，面的信息储存一定要反映出是顺时针方向或反时针方向）。

最后，数据结构将所有的面存储进一个链表或一个数组——具体使用什么结构依赖于应用的需求。如我们将会在下个小节中所看到的，面数据结构将不仅包含顶点的关联列表，而且包含其他的信息，如平面方程的信息，以及用来确定材质属性的相关信息（并最后确定它的颜色）。 169

翼边数据结构

顶点-面数据结构是非常简单的，但是它的能力也是非常有限的。通过在结构中增加边的链表，可以使之更为完善，所以每条边指向顶点链表中的两个入口，然后将面改成由边链表指针来表示。它还可以被进一步完善，让每个顶点维持指向它所属的边和面的指针，边维持指向所属面的指针，等等。它还可以根据特殊需要设计得越来越复杂。

为什么这是重要的呢？原因是这个数据结构可能以不可预知的方式被查询，而且它是进一步构造更丰富的数据结构的一个良好基础，可以让我们在今后根据需要对它作出特别的改变。

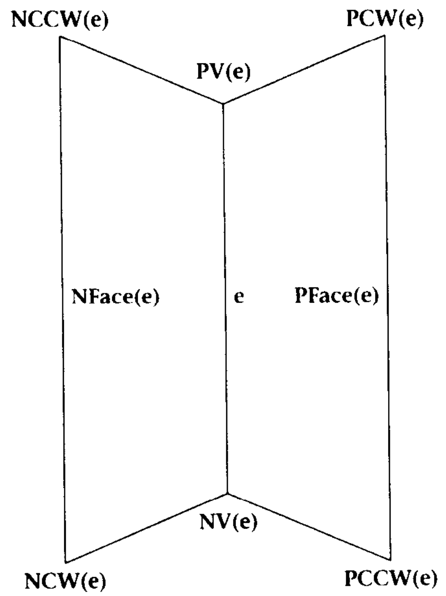
翼边数据结构是由 Baumgart（1975）定义的，它是多面体的一个完备的数据结构，经受住了时间的考验，在应用中得到了广泛的使用。它试图提供一个足够丰富的数据结构，为各种合理查询提供完善的支持。这样的查询的例子如下列各项:

- 对任意面，以顺时针的顺序遍历所有的边;
- 对任意面，遍历所有的顶点;
- 对任意顶点，求出所有与该顶点相交的面;
- 对任意顶点，求出那些与该顶点相交的边;
- 对任意边，求出它的两个顶点;
- 对任意边，求出它的两个面;
- 对任意边，求出在某个面上的下一条边(按顺时针方向或反时针方向)。

图8-4说明了翼边结构的基本思想。对于任意边，总是存在这样的一个排列（已知多面体

的定义)。这是由一组相关的数据结构来表示的。

整个多面体是通过一种骨架来表示的，这个骨架包含了各种环结构（都是双向链表），分别有顶点环、边环和面环。除此之外，骨架还包含到其他骨架的几何关系信息。在顶点环中每个节点包含一个3D 点、到下一个和先前一个顶点的指针，以及到边环的指针。顶点环保存了对象的几何信息。边环保存了对象的拓
170 扑信息——边环还保存有到下一条边和和先前边的指针，还有到所谓的“翼边”结构的指针，以及到它附近顶点和面的指针。翼边数据结构如图8-4所示。面环包含到下一个面和到先前一个面的指针，也包含到边环的指针。



N=下一个
P=前一个
CW=顺时针方向
CCW=反时针方向
V=顶点
e是一条特殊边

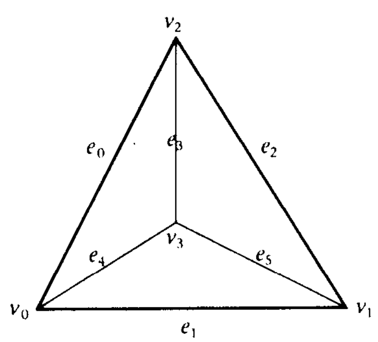
图8-4 翼边数据结构

数据结构的描述用C语言给出（参见附录8.1）。所列出的是一些主要的数据结构，以及一些例子函数。边数据结构是核心——它提供了边和顶点之间的所有连接。顶点数据结构提供的是几何信息。

对象的设计者必须建立所有的主要连接，例如顶点和面之间的连接、边环之间的连接，但是不需计算出所有的翼（nextCWEdge、prevCWEdge、nextCCWEdge、prevCCWEdge)。这些是通过使用在附录中给出的MakeWing子程序自动地构造出来的。对任何两条边，这段子程序能发现它们之间的所有翼边连接。

从头构造翼边结构是相当困难的，且是耗时的过程。实际上，数据结构可以从一个初始的顶点-面表示中构造出来。Chan 和Tan（1988）给出了如何将顶点-面形式的数据转换成翼边形式数据的方法——实际过程要比他们所描述的方法要简单一些。

我们可以将图8-5视为一个例子，这里v_3在这页书
171 所在平面的后面。它的顶点-面数据结构在图下。



顶点	面
v_0	F0=0、3、1
v_1	F1=0、2、3
v_2	F2=3、2、1
v_3	F3=0、1、2

图8-5 一个四面体

现在建立翼边结构，我们需要识别每一条边，而且对每条边确定它的前一个顶点、下一个顶点、前一个面以及下一个面。为了构造边的两个顶点，至于哪个标记为“前一个”，哪个标记为“下一个”完全是任意的。只是一旦选择了，同时哪个为“前一个”面，哪个为“下一个”面因而也就确定了。一旦这些基本连接构造完毕，我们就可以对具有公共顶点的每一对边使用MakeWing算法，因此翼边就形成了。

为了要形成边，只需要遍历每个面，我们知道在面上顶点是按照反时针方向的顺序储存

的。对于一条边，我们将碰到的第一个顶点标记为“前一个”，而把后一个标记为“下一个”。在例子中，我们遇到的第一条边属于面 F0，是从顶点v_0到v_3（在图中，这条边标记为e_4）。对于这条边，依照约定，v_0是“前一个顶点”，v_3是“下一个顶点”。此时，通过对正在遍历的当前面的构造，我们知道（F0）一定是这条边的“前一个面”，这是因为“前一个面”是这样的一个面：当从前一个顶点到下一个顶点移动时，我们是按照反时针方向顺序在移动。

此后，当遍历面F1的时候，我们将再一次遇到这条边，只是现在的顺序是从v_3到v_0。然而，这仍然是同一条边，它不应该被两度存储在边环中。一个新的重要信息是F1一定是这条边的“下一个面”，由此我们可以将这个域添加到该边中。

这样遍历所有面的所有边，我们能够建立起所有需要的初始连接，然后使用MakeWing 172
算法来完成翼边结构（这并不像看起来那么简单，在构造基本元素方面需要非常小心）。

8.4 场景层次结构

基本概念

对象通过数据结构来定义它们的几何性质（顶点）、拓扑性质（边和面间的关系）以及材质属性。对象可以以各种不同的方式被操纵——尤其是平移到另外一个位置，沿着一个轴旋转、伸缩变换，以及这些运算的各种不同的组合形式。这些变换可以通过矩阵变换的使用得以实现，这些已经在第 2 章中讨论过。矩阵变换被应用到对象的顶点上，当然，拓扑关系保持不变。

然而，对象不是孤立地存在的，而是时常相互依赖的——在一个对象或对象某一部分上的变换通常会对其他对象带来一定的影响。在这一小节中我们说明如何建立一个数据结构来表达这些思想，以及如何通过这样的数据结构来实现相互依赖变换的概念。

概念最好是通过举例来说明。考虑图8-6，它给出了一只简单的（二维）机器人的手臂。这个手臂是由数个单元组成的：一个基座、肩关节（S）、上臂（U）、肘关节（E）、前臂（F）和手(H)。自由度的情况是这样的：整个机器人手臂是可以移动的，手臂可以在它的基座上旋转，前臂可以在它的肘上旋转。该图给出了三种情形。在b中，前臂相对于a中的情形旋转了一定的幅度。在c中手臂相对于b中情形在基础上旋转了一定的幅度。

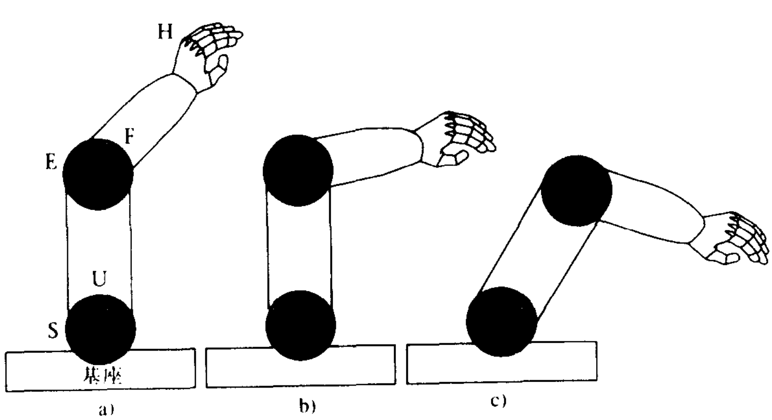


图8-6 一个简单机器人手臂的三个方向

机器人手臂可以被表示为一个简单的层次化模型，如图8-7所示。层次结构指示出了在各个部分之间的关系，这里箭头表示“包含”或
173 “包括”的意义。注意这是一种不对称关系。当基座移动或旋转的时候，在它上面的每个对象（这里只有一个）同样被移动或旋转，而且对象上的每个其他对象本身也同样被转换，在层次结构中处处如此。

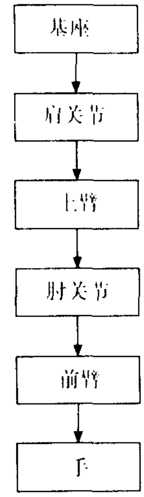


图8-7　机器人手臂的层次化设计

如果在基座上的“肩”关节旋转了，基座本身是不受影响的，但是在肩层次结构中的所有后代是要受到影响的。这里手臂的整体模型是一个层次化数据结构，反映了手臂本身的构造。

我们要把作为手臂模型的数据结构和反映手臂这种数据结构的各种实例图像区分开来。在图形建模中，建模和渲染之间的分离是清晰的，然而也是重要的。建模包括对表现整个对象的数据结构的构造，渲染包括对数据结构的遍历并渲染它的每一个组件。

一般的结构是较为复杂的，不像这里的例子这样的一个简单的线性结构。假设有两个这样的机器人手臂，但是现在它们都是连接到一个躯干的两边，躯干本身又是整个机器人的一部分，机器人有头部、躯干、双臂、双腿等等。机器人的层次化模型如图8-8所示。注意到此时数据结构是一棵树，但是通常它可能是任意无环图。

如果在现实中构造机器人，那么每个元件会被单独制造，并最后全部装配在一起形成机器
174 人。在图形建模中也使用相似的方法。每个组件是在一个适当的空间中单独设计的。举例来说，在简单的机器人手臂情形中，那些关节（包括肘关节和肩关节）都是圆形，可能最初是从坐标空间中的位于原点的一个单位正方形开始的。同样设计者在设计机器人的手时，也是从坐标空间中的一个单位正方形开始的，但他可能喜欢把单位正方形左下角定位在坐标原点上。

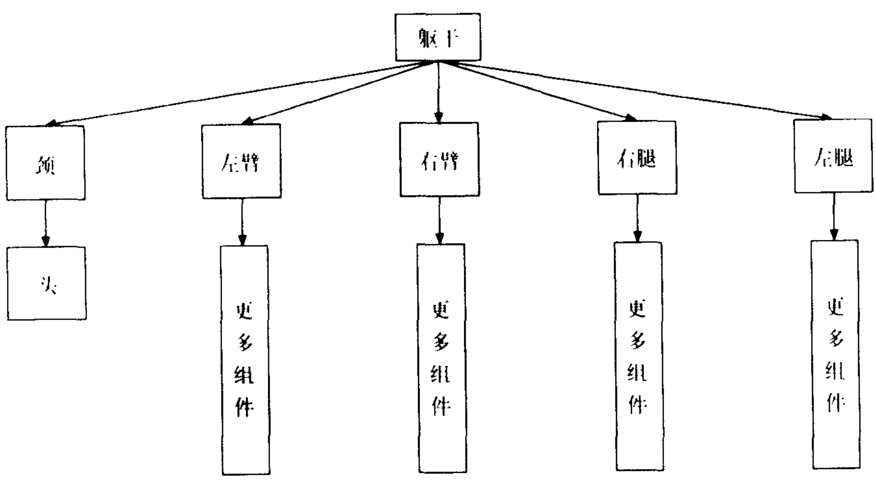


图8-8　机器人的层次化模型

设计组件所使用的坐标系统通常称为*局部坐标系*（LC），有时也叫做*主坐标系*、*建模坐标系*或*对象坐标系*。注意这个LC坐标系是针对每个单独组件的，彼此之间是没有关系的——每个对象都是在它自己的空间中设计完成的。装配是通过产生这些“主对象”的实例来实现的。这包括应用适当的矩阵变换将每个对象定位在相应的位置上，由此建立彼此之间的关联关系。

在图中每个对象明确地用一个节点来表示。在每个节点中有一个对应的变换矩阵，规定了在这个节点中的对象与它的父节点之间的关系。举例来说，对于图8-7中这种情况，无论基座在哪里以及无论它朝向什么方位，基座和肩关节（S）之间的关系可以描述为一个变换矩阵(相对于描述基座的那个坐标系统)。同样地，上臂（U）对S的关系也能用一个变换矩阵来描述。因此在层次结构中对象之间的关系总是能通过这样的矩阵得以描述。由此可见，在任何节点上的对象和它的祖先之间的关系，也就是说与层次结构中该对象所在节点之上的各层节点对象之间的关系都能通过一个单一变换矩阵来描述，这是通过路径上的各矩阵相乘得到的矩阵来实现的。

机器人手臂移动和操作的仿真是通过在各种不同层次上变换相应的变换矩阵来达到的。举例来说，整个手臂移动的仿真就可以对基座相应的变换矩阵进行变换来达到。模拟前臂的 175
旋转只要改变前臂（或肘）相关联的变换矩阵就行了。因此模型的一个实例就是一组特殊变换矩阵的集合。

当我们渲染模型的时候，从层次结构的根开始对整个结构进行遍历。与根相关联的矩阵可以看成是在WC坐标系中变换根对象。先执行这个变换再对根对象进行渲染。根的每个孩子也根据它们相关联的矩阵进行相应的变换。这个变换是将这些对象相对于根的局部坐标系进行的相对变换。经过转换的对象然后还要根据根节点相关联的变换矩阵进行变换(这是在WC坐标系内的变换)，然后渲染每个对象。这个相同的过程在层次结构中各节点处执行，路径上的矩阵乘积构成了对象在 WC 中的相应变换。

假设与基座、肩关节、上臂、肘关节、前臂以及手相关联的矩阵分别是 B、S、U、E、F和H。假设b是基座上的一个点(在基座的局部坐标系中描述的)。那么 $b.B$ 是在WC中描述的。假设有：s、u、e、f和h分别是肩关节等对象局部坐标系中的点，那么图8-9给出了在WC中的这些点。

b.B
s.SB
u.USB
e.EUSB
f.FEUSB
h.HFEUSB

图8-9 变换的层次结构

所以要注意，在这个例子中，矩阵F的改变只会影响前臂和手的渲染。矩阵B的改变将会影响整个手臂，而矩阵H的改变只会影响到手。虽然这个例子是简单的线性层次结构，但是它同样适用于一般的无环图。

从上述讨论中我们可以清楚地看出，建模是渲染之前的一个阶段。对象首先在自己的局部坐标系中被建模（LC）。当遍历层次结构时建模变换就产生对象在 WC 坐标系中的位置，然后被送进渲染系统。

与对象关联的矩阵

现在考虑下面这张表，它给出了与每个组件对象相关联的各个不同矩阵。

对象	局部变换矩阵（LTM）	当前变换矩阵（CTM）	全局变换矩阵	
基座	*B*	*B*	*I*	
肩关节	*S*	*SB*	*B*	
上臂	*U*	*USB*	*SB*	176
肘关节	*E*	*EUSB*	*USB*	
前臂	*F*	*FEUSB*	*EUSB*	
手	*H*	*HFEUSB*	*FEUSB*	

当前变换矩阵（CTM）将一个点从LC坐标系转换到WC坐标系。它是通过将直达根节点的路径上的LTM相乘得到的。全局变换矩阵（GTM）主要是为计算CTM提供方便。在任何一个节点上，GTM等于其父节点上的CTM。因此有关系：

$$CTM=LTM \times GTM \tag{8-11}$$

从这个角度看WC概念变成了一个相对的概念，认识到这一点是非常重要的。WC是因时而变的，它随着所要渲染的层次结构的根的变化而变化。当然，这个层次结构还可能是更大的一棵树的一个子树。想像有一个包含各种家具和墙壁上挂着一幅照片的房间。房间整体上可能作为层次结构的根被渲染，这就是这种意义上的“世界坐标”。然而，如果墙壁上的照片相应的子树是渲染的对象的话，那么现在这就是WC坐标系。另一方面，如果房间在一幢房子里面，整个房子作为渲染对象，那么房子就成为WC坐标系，如此等等。

数据结构

对象的基本描述方法一般是将其描述为多边形的集合，例如翼边结构。我们把这些称为基本描述结构。举例来说，有一个特别结构可能是一书桌的局部坐标描述。这个结构可以被层次结构中的一个或者多个对象使用（“被实例化”）。最后这些结构中多边形的顶点坐标要经过与变换矩阵相乘转换到WC坐标系。对象层次结构的数据结构如下所示：

```
struct Object   {
   WingedEdge structure; /*representation of LC polys*/
   Matrix  CTM, LTM, GTM;
   int  noOfChildren;
   Object  child[]; /*array of children*/
}
```

那么，为了遍历层次结构中的某个特定子树，我们就要对每个节点使用函数 f()：

```
traverseObject(Object object, Function f) {
177      f(object);
      if(object.noOfChildren > 0){
         for(i=0; i< object.noOfChildren;++i){
            traverseObject(object.child[i],f);
         }
      }
   }
}
```

尤其是，f()包括用CTM乘以结构中的多边形顶点，因此将它们转换到WC坐标系，然后再进行渲染。

下面两个函数设定对象间正确的矩阵连接。

```
setGlobalMatrixOfObject(Object object, Matrix m)  {
   object.GTM = m;
   /* The "*" represents matrix multiplication*/
   object.CTM = object.LTM*object.GTM;
   if(object.noOfChildren > 0)  {
      for(i=0; i< object.noOfChildren;++i){
         setGlobalMatrixOfObject(object.child[i], object.CTM);
      }
   }
}

setLocalMatrixOfObject(Object object, Matrix m)  {
```

```
    object.LTM = m;
    object.CTM = object.LTM*object.GTM;
    if(object.noOfChildren > 0){
      for(i=0; i< object.noOfChildren;++i){
        setGlobalMatrixOfObject(object.child[i], object.CTM);
      }
    }
}
```

为了使用对象层次结构描述一个场景，我们需要做以下几件事情:

（1）建立对象在场景中的层次关系——即建立连接。

（2）设计每个结构，即为每个对象选择一个适当的坐标系并构造对象。

（3）利用 LTM 建立每个子节点和它的父节点之间的几何关系，LTM将子节点的坐标转换为父节点所在的坐标系统。

（4）调用函数setLTM()以便为每个对象设定 LTM。

8.5 使用 OpenGL

OpenGL是一种渲染系统。它没有用特殊的数据结构来表达多面体。它提供了对层次对象模型构造的支持，但是没有给出具体实施的特别策略。OpenGL 提供了渲染图形简单实体（多边形）的能力，以及在渲染之前应用矩阵变换到多边形顶点的能力。在下一章中我们将会更具体地分析这些是如何实现的，即在完成对象层次结构构造之后，如何使用OpenGL 来有效地进行渲染。 178

在这里我们用例子说明在OpenGL中如何直接使用模型视图矩阵堆栈来产生对象层次结构。这里所使用的方法还很不理想，但是理解这个例子将会对了解模型视图矩阵堆栈的工作原理有帮助。

代码的目的是创建一系列立方体并将它们依次罗列在一起。一个立方体位于前一个立方体的上面，上面的一个立方体在每个维度上都比下面的一个缩小一半。这里创建了一个立方体的实例，而且它的六个面分别是一个面的六个拷贝，通过旋转和平移将它们组合在适当的位置上所构成的。

在函数 cubebase中，我们创建了一个正方形，它的两个对角坐标分别是（-0.5，-0.5，0.0）和（0.5，0.5，0.0）。这是所创建立方体的基础。

在函数cube中单个面被调用六次，每次所使用的都是一个新的矩阵变换。该函数中第一个调用（glMatrixMode）设置了模型视图矩阵堆栈，此后还要在后续的调用中对这个模型视图堆栈进行操作。第二个函数（glPushMatrix）将当前的模型视图矩阵的一个拷贝压入堆栈。这是将后续的建模变换与视图变换隔离开来（在下一个章中我们再考虑它）。然后是对变换函数（glTranslate和glRotate）的成功调用。第一个调用（“d”——表示所使用的是双精度型）构造了一个平移矩阵，并用它预乘当前的模型视图矩阵。举例来说，如果C是当前的模型视图矩阵，T 是平移矩阵，在调用平移函数之后新的模型视图矩阵是TC。

glRotate（a，x，y，z）构造了一个旋转矩阵，它的含义是绕从原点到点（x，y，z）的矢量反时针旋转a角度。假设这个矩阵是R，那么新的模型视图矩阵是 RC。因此调用序列是:

```
glTranslated(a,b,c);/*matrix T*/
glRotated(theta,x,y,z); /*matrix R*/
```

由此得到的新的模型视图矩阵是 RTC（注意矩阵是按所定义的顺序反序作用于变换的）。

函数cube因此使用最初的基构造了立方体的所有六个面。

现在子程序stack定义了一个立方体，然后递归调用它n次，每一次后续的调用都装入了一个新的立方体，这个新立方体在每个维度上都较上一个缩小一半，并被平移到前一个立方体的上面。当这个过程结束时我们就已经定义了一个立方体的堆栈。

值得再次强调的是，这并不是构造对象层次结构的一个精细的或高效的办法。这个方法
179 太特别、也太基本，直接依赖于 OpenGL 矩阵设施。更适当的方式应该如我们在之前所讨论的那样，构造一个对象数据结构。然后使用OpenGL设施渲染这个层次结构，这将在下一章中讨论。

```
static void cubebase(void)
/*specifies a side of a cube*/
{
  glBegin(GL_POLYGON);
    glVertex3d(-0.5,-0.5,0.0);
    glVertex3d(-0.5,0.5,0.0);
    glVertex3d(0.5,0.5,0.0);
    glVertex3d(0.5,-0.5,0.0);
  glEnd();
}

static void cube(void)
/*uses cube side to construct a cube, making use of the modelview
matrix*/
{
  /*make sure we're dealing with modelview matrix*/
  glMatrixMode(GL_MODELVIEW);

  /*pushes and duplicates current matrix*/
  glPushMatrix();

  /*construct the base*/
  cubebase();

  glPushMatrix();
  /*construct side on +x axis*/
  glTranslated(0.5,0.0,0.5);
  glRotated(90.0,0.0,1.0,0.0);
  cubebase();

  glPopMatrix();

  /*construct side on -x axis*/
  glPushMatrix();
  glTranslated(-0.5,0.0,0.5);
  glRotated(-90.0,0.0,1.0,0.0);
  cubebase();
  glPopMatrix();

  /*construct side on +y axis*/
  glPushMatrix();
  glTranslated(0.0,0.5,0.5);
  glRotated(-90.0,1.0,0.0,0.0);
  cubebase();
  glPopMatrix();
```

180

```
  /*construct side on -y axis*/
  glPushMatrix();
  glTranslated(0.0,-0.5,0.5);
```

```
    glRotated(90.0,1.0,0.0,0.0);
    cubebase();
    glPopMatrix();

    /*construct top*/
    glPushMatrix();
    glTranslated(0.0,0.0,1.0);
    glRotated(180.0,1.0,0.0,0.0);
    cubebase();
    glPopMatrix();

    glPopMatrix();

}

static void stack(int n)
/*creates a smaller cube on top of larger one*/
{

    cube();
    if(n==0)return;

    glPushMatrix();
    glTranslated(0.0,0.0,1.0);
    glScaled(0.5,0.5,0.5);
    stack(n-1);
    glPopMatrix();
}
```

8.6 使用VRML97

VRML97对多面体的定义方法与上面描述的方法十分类似。虽然一些基本实体是已经定义好了的，可以被直接使用外，绝大多数的几何体是使用IndexedFaceSet节点来定义的，它采用的是顶点-面数据结构。下列例子定义了一个棱柱和一个立方体（我们暂且忽略内嵌的Box节点），说明一个对象层次结构是如何建造的，并介绍VRML方法。

一个VRML场景图是节点的分层集合。场景图的语义不只是一个变换层次结构，而且我们可以将一个Group节点看成是一个压入堆栈的单位矩阵，所以它在场景图中构成了将部分节点与其他部分节点的逻辑分离。

在场景中的可见对象是用Shape节点定义的。一个Shape节点有两个域：外观和几何。这些域本身也是节点。第一个是Appearance节点。第二个是几何节点中的一种，而且在这里它就是IndexedFaceSet。在图8-10的例子中，所定义的第一个对象是立方体。语句DEFmy_box将这部分场景图作标记待稍后引用（见下面）。立方体的IndexedFaceSet节点包含了对顶点的描述，这些顶点存在于coord域中（它本身是一个坐标节点），Indexed 181
FaceSet节点还包含了到coordIndex域中顶点的一个指针链表。每个面是一串顶点，用-1作为结束标记。作为缺省约定，面应该以反时针方向的顺序定义。

第二个可见对象为棱柱，它是放置在一个方盒上的，所以我们把它封装在Transform节点中。变换可以是平移、旋转和伸缩，它们被应用到场景图中更低层中的对象上。我们又一次使用了一个IndexedFaceSet节点，这次我们用my_prism来标记它，并带有一个关键词DEF。

```
#VRML V2.0 utf8
Group {
 children [
  Shape {# The first box
   appearance Appearance {
    material Material {# A green material
     diffuseColor 0.1 0.7 0.2
     } }
   # Define the geometry for a box
     geometry DEF my_box IndexedFaceSet {
      coord Coordinate {
       point [-1 0 -1,1 0 -1,-1 0 1,1 0 1,
              -1 2 -1,1 2 -1, -1 2 1, 1 2 1]
      }
      coordIndex [0,1,3,2,-1,1,5,7,3,-1,
                  2,3,7,6,-1,0,2,6,4,-1,
                  0,4,5,1,-1,4,6,7,5,-1]
  } }
#Position the first prism on the first box
 Transform {
  translation -0.5 2 0
   children [
    Shape {#The first prism
     appearance Appearance {
#A reddish material
      material Material {
       diffuseColor 0.5 0.2 0.2
      } }
#Define the geometry for a prism
    geometry DEF my_prism IndexedFaceSet {
     coord Coordinate {
      point [-0.5 0 0.5, 0.5 0 0.5, 0.5 0 -0.5,
             0 1 -0.5, 0 1 0.5,-0.5 0 -0.5]
     }
      coordIndex [0,1,4,-1,5,3,2,-1,
                  1,2,3,4,-1,0,5,2,1,-1,
                  3,5,0,4,-1]
    }
   }
  ]
 }
#Position the second box on the first box
 Transform {
  translation 0.5 2 -0.5
  scale 0.25 0.25 0.25
  children [
   Shape {# The second box
    appearance Appearance {
     material Material {# A blue material
      diffuseColor 0.3 0.4 0.8
     }
    }

#"Instance" the box geometry
    geometry USE my_box
   }

#Position the second prism on the second box
 Transform {
  translation 0 2 0
  rotation 0 1 0 0.7853
   children [
    Shape {# The second prism
     appearance Appearance {
      material Material {
       diffuseColor 0.7 0.7 0.7
      }
     }

#"Instance" the prism geometry
      geometry USE my_prism
     }
    ]
   }
  ]
 }
]
}
```

182

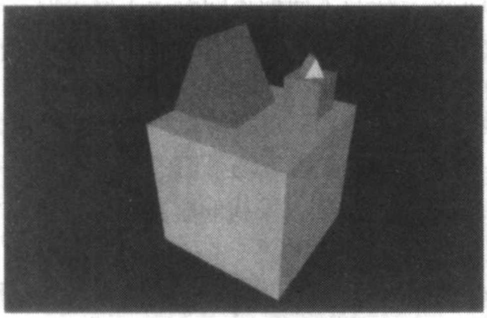


图8-10 VRML97的示例场景

第三个可见对象是另外一个立方体。它与棱柱在层次结构中同一个层次上，也就是说，它不是棱柱的子对象，但它是第一个立方体的子对象。我们将这个方盒在每个方向上都乘以0.25，并将它平移到第一个立方体上面的一个角上。因为我们已经定义了立方体的一些几何特征，我们可以仅仅通过写USE my_box来完成对它的实例化。在VRML中实例化是对场景图中的另外一部分的一个引用，而不是对它的一个完全拷贝。这种机制既可以减少对场景图存储的内存耗费，也增进了系统的性能，因为共享的几何特征可以被渲染系统进一步优化和多次复用。

第四个可见对象是第二个棱柱。这是第二个立方体的一个子对象，它位于这个立方体的上面，并被旋转了π/4个弧度。关于对它的变换有两点需要注意。因为它是第二个立方体的一

个子对象，它只有第一个棱柱的四分之一大小，这是因为当前变换矩阵仍然反映了先前的尺寸。还需要注意的是，这并不意味着将第二个棱柱放于第二个立方体上的平移变换不同于将第一个棱柱放于第一个立方体上的平移变换，因为平移处理是相对于当前局部坐标系统的。在局部坐标系中，第二个立方体仍然是两个单位高，所以我们一定要将第二个棱柱向上移动两个单位。

8.7 小结

在这一章中我们详细地讨论了多边形，也讨论了对象和场景创造的问题，包括多面体表示的数据结构、对包含多个多面体的复杂对象描述的层次化数据结构。实际上整个场景可以被看成是一个复杂“对象”，或者看成是整个对象层次结构的根。我们看到，对层次结构的创造不但包括对子节点与父节点之间关系“连接”的创建，而且还要创建把对象转换到其父节点坐标系的变换矩阵。我们定义了“局部变换矩阵”和“当前变换矩阵”，它们将对象的顶点转变到层次结构根节点对象所在的坐标系（“世界坐标”）中。最后，我们简单介绍了在两个广泛使用的系统OpenGL和VRML中是如何创建这样的层次结构的。 183

有关本章内容的一个要点是场景描述独立于观察的思想：观察和建模是独立（尽管OpenGL中有“模型视图”矩阵这个合成概念）。我们首先创建一个场景，然后可以选择那个场景的一个任意视图。这将是下一章的主题。

附录8.1 翼边数据结构的C语言描述

```
/*winged edge data structure for representing polyhedra*/

#define NOWINGEDEDGE ((WingedEdge *)0)
#define NOVERTEXELEMENT ((VertexElement*)0)
#define NOEDGEELEMENT ((EdgeElement *)0)
#define NOFACEELEMENT ((FaceElement *)0)

typedef struct _vertexElement{
  Point3D *p;
  struct _vertexElement *nextVertexElement, *prevVertexElement;
  struct _edgeElement *edgeElement;
} VertexElement, *VertexElementPtr;

typedef struct _faceElement{
  Face *face;
  struct _faceElement *nextFaceElement, *prevFaceElement;
  struct _edgeElement *edgeElement;
} FaceElement, *FaceElementPtr;

typedef struct _edgeElement {
  struct _edgeElement *nextEdgeElement, *prevEdgeElement;
  VertexElement *nextVertexElement, *prevVertexElement;
  FaceElement *nextFaceElement, *prevFaceElement;
  struct _edgeElement *nextCWEdgeElement, *prevCWEdgeElement,
                      *nextCCWEdgeElement, *prevCCWEdgeElement;
} EdgeElement, *EdgeElementPtr;

typedef struct{
  VertexElementPtr vertexElement;
  FaceElementPtr faceElement;
```

```
    EdgeElementPtr edgeElement;
} WingedEdge, *WingedEdgePtr;

static void makeWings(EdgeElement *e1, EdgeElement *e2)
/*given two edges, this function finds all the wings*/
{
    VertexElement *e1pv, *e1nv, *e2pv, *e2nv;
    FaceElement *e1pf, *e1nf, *e2pf, *e2nf;

    if(e1==NOEDGEELEMENT || e2==NOEDGEELEMENT) return;

    e1pv = e1->prevVertexElement;
    e1nv = e1->nextVertexElement;
    e2pv = e2->prevVertexElement;
    e2nv = e2->nextVertexElement;
    e1pf = e1->prevFaceElement;
    e1nf = e1->nextFaceElement;
    e2pf = e2->prevFaceElement;
    e2nf = e2->nextFaceElement;

    if((e1pv==e2pv) && (e1pf==e2nf) ) {
        e1->prevCWEdgeElement = e2;
        e2->nextCCWEdgeElement = e1;
        return;
    }

    if((e1pv==e2pv) && (e1nf == e2pf)   ) {
    e1->nextCCWEdgeElement = e2;
        e2->prevCWEdgeElement = e1;
        return;
    }

    if((e1pv==e2nv) && (e1pf == e2pf)   ) {
        e1->prevCWEdgeElement = e2;
        e2->prevCCWEdgeElement = e1;
        return;
    }

    if((e1pv==e2nv) && (e1nf == e2nf)   ) {
        e1->nextCCWEdgeElement = e2;
        e2->nextCWEdgeElement = e1;
        return;
    }

    if((e1nv==e2pv) && (e1pf == e2pf)   ) {
        e1->prevCCWEdgeElement = e2;
        e2->prevCWEdgeElement = e1;
        return;
    }

    if((e1nv==e2pv) && (e1nf == e2nf)   ) {
        e1->nextCWEdgeElement = e2;
        e2->nextCCWEdgeElement = e1;
        return;
    }

    if((e1nv==e2nv) && (e1pf == e2nf)   ) {
        e1->prevCCWEdgeElement = e2;
        e2->nextCWEdgeElement = e1;
        return;
```

184

```
    }

    if((e1nv==e2nv) && (e1nf == e2pf)  ) {
      e1->nextCWEdgeElement = e2;
      e2->prevCCWEdgeElement = e1;
      return;
    }
}
static short commonVertex(EdgeElement *e1, EdgeElement *e2)
/*returns 1 if edges share a vertex*/
{
   return(e1->prevVertexElement==e2->prevVertexElement) ||
         (e1->prevVertexElement==e2->nextVertexElement) ||
         (e1->nextVertexElement==e2->prevVertexElement) ||
         (e1->nextVertexElement==e2->nextVertexElement);
}

static void makeAllWings(EdgeElement *edgeElement)
{
   EdgeElement *e1, *e2;

   e1 = edgeElement;
   do{                                                        185
     e2 = e1->nextEdgeElement;
     do{
       if(commonVertex(e1,e2)) {
         makeWings(e1,e2);
       }
       e2 = e2->nextEdgeElement;
     } while(e2 != edgeElement);
     e1 = e1->nextEdgeElement;
   } while(e1 != edgeElement);
}

EdgeElement *ccwEdgeAfterEdge(EdgeElement *edgeEl, FaceElement
*faceEl)
/*returns the next CCW edge belonging to the given face after this
edge*/
{
   if(edgeEl->prevFaceElement == faceEl) return(edgeEl-
   >prevCCWEdgeElement);
   else
   if(edgeEl->nextFaceElement == faceEl) return(edgeEl-
   >nextCCWEdgeElement);
}

EdgeElement *cwEdgeAfterEdge(EdgeElement *edgeEl, FaceElement
*faceEl)
/*returns the next CW edge belonging to the given face after this
edge*/
{
   if(edgeEl->prevFaceElement == faceEl) return(edgeEl-
   >prevCWEdgeElement);                                      186
   else
   if(edgeEl->nextFaceElement == faceEl) return(edgeEl-
   >nextCWEdgeElement);
}
EdgeElement *ccwEdgeAfterVertex(VertexElement *vel, FaceElement
*faceEl)
/*returns the next CCW edge belonging to the given face after this
```

```
vertex*/
{
  EdgeElement *edge;

  /*get the edge for the vertex*/
  edge = vel->edgeElement;
  /*this will always be such that vel is the nextV for this edge*/

  if(faceEl == edge->nextFaceElement) return edge;
  else
  if(faceEl == edge->prevFaceElement) return edge-
  >prevCCWEdgeElement;
  else
  return edge->nextCWEdgeElement;
}

EdgeElement *cwEdgeAfterVertex(VertexElement *vel, FaceElement
*faceEl)
/*returns the next CW edge belonging to the given face after this
vertex*/
{
  EdgeElement *edge;

  /*get the edge for the vertex*/
  edge = vel->edgeElement;
  /*this will always be such that vel is the nextV for this edge*/

  if(faceEl == edge->nextFaceElement) return edge->nextCWEdgeElement;
  else
  if(faceEl == edge->prevFaceElement) return edge;
  else
  return edge->prevCCWEdgeElement;
}

VertexElement *ccwVertexAfterVertex(VertexElement *vel, FaceElement
*faceEl)
/*returns the next CCW vertex belonging to the given face after this
vertex*/
{
  EdgeElement *edge;
  VertexElement *v;

  /*get the next CCW edge*/
  edge = ccwEdgeAfterVertex(vel,faceEl);

187  if((v=edge->prevVertexElement)==vel) return edge-
  >nextVertexElement;
  else return v;
}
VertexElement *cwVertexAfterVertex(VertexElement *vel, FaceElement
*faceEl)
/*returns the next CW vertex belonging to the given face after this
vertex*/
{
  EdgeElement *edge;
  VertexElement *v;

  /*get the next CCW edge*/
  edge = cwEdgeAfterVertex(vel,faceEl);

  if((v=edge->prevVertexElement)==vel) return edge-
```

```
  >nextVertexElement;
  else return v;
}

void applyCCWEdgesOfFace(FaceElement *faceEl, void (*f)(EdgeElement
*edge))
/*runs through all edges of face in CCW order, applying function f*/
{
  EdgeElement *e0,*edgeEl;

  /*get any edge of this face*/
  e0 = edgeEl = faceEl->edgeElement;

  do{
    (*f)(edgeEl);
    edgeEl = ccwEdgeAfterEdge(edgeEl,faceEl);
  } while(edgeEl != e0);
}
```

188

第9章 投影：照相机模型的实现

9.1 引言

在第7章中我们说明了如何去构造一个照相机，使得我们可以从任意视点和任意方向观察场景。我们接着又说明了如何扩充场景，使得场景中不光只有球体，还可以包括多面体。多面体是构成场景描述的基本组件块。在第8章讨论了光线跟踪多边形之后，我们介绍了使用光线跟踪对相当复杂场景的定义和渲染的一些工具。然而在那里我们还是存在一个问题没有解决。这些场景越是丰富和有趣，我们就需要越长的时间来完成对它们的渲染：从几分钟到数天。

在这一章中我们将开始一个战略转变过程，从通过光线跟踪所提供的光照真实感转向实时解决方案的实现。为什么光线跟踪要花费那么长的时间？如我们早些时候所了解到的，这是因为存在着大量的相交计算。原则上对每条光线都一定要计算出它与每个对象之间的相交情
189 况，以便找到最近处的相交点，如果有的话。这个计算量实际上是可以大大减少的，如我们将在第16章中所看见的那样。但是即使它被减少了，仍然还会有大量的光线需要处理，因为除了从 COP 经过像素的那些最初主光线之外，还有衍生出来的反射光线和传导光线。

那么我们能做的第一件事就是降低反射和传导需求的光线数量。换句话说，我们只计算从光源直接照射对象表面所产生的光照效果，而不考虑在对象之间的相互反射所产生的光线。这将在很大程度上降低真实感。

我们仍然还要处理主光线。主光线仍然必须与场景中每个对象进行相交计算，以便比较哪个的交点是最近的一个交点。这仍然是十分慢的，几乎不可能达到实时的性能要求。举例来说，每一次移动照相机，所有主光线都必须重新发射，而当每一个对象移动的时候，至少主光线的一个子集必须重新考虑。

所以加速计算的第二个方法就是要放弃从眼睛向场景发射光线的整个思想。

换个思考方式，我们可以更加快速地去渲染场景。让我们回想一下先前对每个多边形预定一个颜色的做法（也就是说，忘掉光照计算）。如果你通过光线投射将这样一个多边形投影到视图平面上，结果将会是视图平面上的一个多边形。这与经过每个多边形顶点和COP点的光线与视图平面相交所形成的多边形是一致的。在视图上所投影的顶点构成了视图平面上的一个多边形（回忆一下关于仿射变换的讨论）。在视图平面上的多边形用的是多边形预定颜色做的明暗处理。这种速度上的改变是巨大的。先前需要从每个像素开始做光线投射，并且要去求与多边形的相交。这里我们仅仅需要计算很少量的光线和视图平面的相交——对每个顶点进行一次，通过上面的构造，我们就确切知道与多边形相交了。

若把自己限制在场景整个都是用多面体构造的范围内，那么我们的新方法可以描述如下（如图9-1）：

（1）投影多边形到在视图平面上。对任何多边形，求出经过COP点和多边形顶点的光线在视图平面上的所有交点。这在视图平面上定义了一个二维多边形。把投影顶点（二维顶点）转换到显示器坐标，并在二维中用所需的明暗效果来绘制多边形。

（2）对每个多边形重复上面的过程。

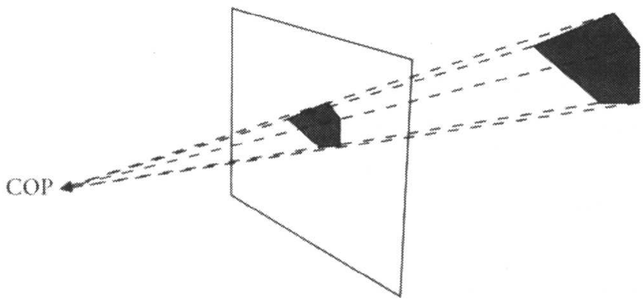


图9-1 将多边形投影到视平面并渲染它 190

不同于光线跟踪的数以百万计的光线数量，我们这里对于整个复杂场景来说也大约只有数万个光线-视图平面交点。而且，没有牵涉到“跟踪”问题——每条光线都是来自于一个对象的（因为每条光线经过对象所定义的顶点）。

当然，在采用这个方法时我们也要介绍一些额外的问题：

- 光线跟踪隐式地为我们解决了“可见性”问题。在新的方法中我们仅仅有一组多边形。如何解决它们之间的可见性关系？这是一个主要问题。
- 光线跟踪是与光照模型一块使用的。现在我们像是在渲染二维多边形。光照效果该如何被重新导入呢？
- 给定二维上的一个多边形，我们如何高效率地确定组成它的像素集合？换句话说，我们如何渲染一个二维多边形？
- 光线跟踪隐式地解决了“视景体可见性”问题：处于视景体外面的光线是不会被生成的。现在对于新方法这种情况不再成立。位于视景体外面的多边形要被忽略掉，部分在视景体内的对象需要修整，只留下在视景体内部的那部分。这个过程称为“裁剪”。

我们看起来像是在某些方面有所改进，而在另一些方面有所退步。接下来几章的目的是要说明这些问题中每一个是如何得以解决的，从而使得我们能够实现一个高效率的“渲染管道”。在这一章中我们将集中注意力于在视图平面上多边形的投影过程。在此之后，我们考虑将在稍后的各章中讨论前面所提出的各种问题。

9.2 完整的照相机描述

在进一步展开讨论之前，我们必须完成对虚拟照相机所有参数的完整描述。至此已经定义了VRP、VPN、VUN 和 COP。除此之外，还有视平面窗口。我们还需要定义更多内容。 191

有时我们对照相机模型中“眼睛”的角色有些模糊，眼睛是什么？我们此处将它称为眼睛是否也存在一点问题——举例来说，COP 有时就是指“眼睛”。然而，在现实生活中我们使用眼睛来定位照相机，使之对准一个真实场景来捕获图像。然后通过我们的眼睛去看相片上的图像。对于这种抽象照相机情况，我们书写一段程序，让它相对于抽象场景对视平面定位和定向以及对投影中心定位。我们知道场景以对象数据库的形式“存在”，如在第 8 章所描述的那样。一旦这个场景被投影到视平面上，并在显示器上得到渲染，我们就开始使用我们的真实眼睛去看它了。这里我们介绍定义照相机所需要的函数。在确定了观察坐标系后，使用VRP、VPN和VUV就可以定义剩余的这些参数了，它们都是在VC中定义的：

视平面距离。视平面是一个与VPN垂直的平面，场景被投影到此平面上。视平面距离指从原点开始沿着VPN方向到视平面的距离（即从观察坐标系的原点、沿着这个坐标系的 N 轴方向）。

投影类型。有两个主要的投影类型，分别是*透视投影*和*平行投影*。对于透视投影这种情况，来自场景每个点的光线汇聚到一个特殊点——*投影中心*（COP）。这些光线与视平面的交点构成投影。COP点定义为相对于VRP的偏移（即作为观察坐标系中的一个点）。平行投影的构造是从场景中的点开始做平行光线，它们与视平面的交点所构成的投影。这些平行光线的方向就称为投影的方向（DOP）。正交平行投影的方向是（0，0，−1），也就是说，光线与 N 轴平行。概念上平行投影可以被认为是 COP 在“负无穷远”处。

对于透视投影，当场景中的平行线不与视平面平行时，它们将会在投影图像中聚合于一个*灭点*上。与主轴平行的各直线将聚合到一个*主灭点*上。在一个投影中最多有三个主灭点。主灭点的数目等于与视平面相交的主轴数。有时人们由此区别这些透视投影“类”，分别用1点、2点和 3点透视来形容这些特别类型。*等角投影*是这样一种投影，它的视平面与每个主轴的夹角都相等。

192 在平行投影中，如我们前面所看到的，正交投影的投影方向是与视平面的法向一致的。对于斜的平行投影不成立。Carlbom 和 Paciorek（1978） 对投影类型有一个完整的描述。

视平面窗口。这是视平面上的一个窗口——定义为一个矩形，它的边分别与 V 轴和 U 轴平行。对于透视投影，从 COP 点出发经过这个窗口四个顶点的光线确定了视景体，这是一个双无限棱锥（可能不规则）。视景体类似于视觉圆锥体，它只包括那些从这个特殊视点所能看到的场景。对于平行投影也有一个相似的构造（在后面的进一步讨论中我们只考虑透视投影的情况）。显然视景体的作用是一个 3D 裁剪区域。

前裁剪平面和后裁剪平面。至此为止的构造实际上并未排除位于 COP 后面的场景中的对象。除此之外，观察者可能希望排除场景中那些离COP点“太近”和“太远”的部分。排除后者是为了提高渲染效率（不去渲染那些位于极远处、对图像没有什么贡献的对象），排除前者是为了避免渲染那些位于COP后面的对象以及避免数值的不稳定。这可以通过定义前裁剪平面和后裁剪平面来实现。这些平面是与视平面平行的平面，它们被安放在和 VRP 距离为某个确定值的位置上。

这一小节的思想将用图9-2来说明。

为了要在程序设计语言中表达这些思想，我们构造一个称为`Camera`的数据结构，用它来包含各种不同参数设置实例。

```
typedef struct{
   /*camera parameters and implementation information*/
   /*...*/
} Camera;

Camera *newCamera(void);
/*creates a new camera for a perspective view*/

void setVRP(Camera *camera, double x, double y, double z);
/*sets the View Reference Point*/

void setVPN(Camera *camera, double x, double y, double z);
/*sets the View Plane Normal*/
```

```
void setVUV(Camera *camera, double x, double y, double z);
/*sets the View Up Vector*/

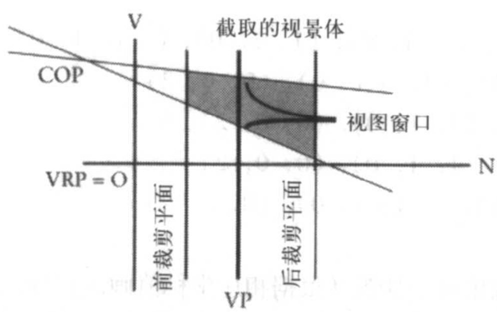
void setCOP(Camera *camera, double x, double y, double z);
/*sets the Centre of Projection*/

void setVPWindow(Camera *camera, double xmin, double xmax, double
ymin, double ymax);
/*sets the View Plane Window*/

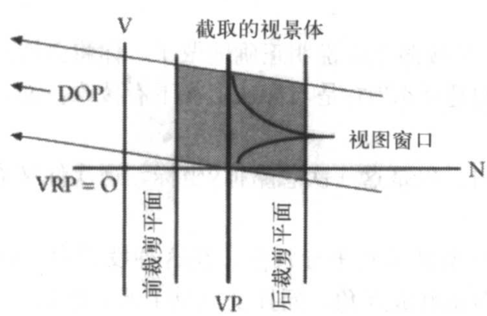
void setVPDistance(Camera *camera, double vpd);
/*sets the View Plane Distance*/
void setClipPlanes(Camera *camera, double front, double back);
/*sets the front and back clipping planes*/

void clickView(Camera *camera);
/*does the transformation to viewing coordinates: using VRP, VPN, VUV.
Transforms to a RHS looking down negative z-axis*/
```

193



a）透视投影



b）平行投影

图 9-2

9.3 投影

投影是将一个三维场景在一个二维平面上表现出来的过程。人类早在史前时期就学会了在二维平面上表现真实世界；我们每个人都知道洞穴绘画。然而，在漫长的人类历史中，只

直到了最近我们才清楚在二维平面上正确表现三维场景的原理。这是在文艺复兴时期发现的。

194 问题是：给定一个三维对象，我们如何在一个二维平面上表现它？答案由艺术家和数学家共同完成，这就是投影的概念。投影是将三维空间中的每个点与二维空间上的一个特定点建立关联关系，从而通过这个二维平面来表现三维场景。注意这总是一对多的关系，也就是说，在二维空间中的一个点一般将会表示三维空间中无穷数目个空间点。

投影有两个基本方式，如上面所讨论的。第一个是平行投影，第二个是透视投影。进而又有许多子类型。

平行投影

我们首先分析平行投影和下列各个例子（读者应该独立完成下列各项内容以加深对这些内容的理解）。

下列点序列在三维空间中定义了一个简单的棱锥形状。我们将给出棱锥的每条边和面的坐标：

底：（－1，－1，0），（1，－1，0），（1，1，0），（－1，1，0）

侧面1：（－1，－1，0），（1，－1，0），（0，0，2）

侧面2：（1，－1，0），（1，1，0），（0，0，2）

侧面3：（1，1，0），（－1，1，0），（0，0，2）

侧面4：（－1，－1，0），（－1，1，0），（0，0，2）

执行下列步骤：

（1）使用3D坐标系画出这个棱锥（根据相应坐标值画出并标记出棱锥的轴，然后插入棱锥的侧面的每个棱边。注意这个坐标系，包括你所画的内容，都将是投影。除非书变成了一种全息图，或者是在虚拟现实中，否则都是这样）。

（2）现在让我们重新在一个二维(x，y)坐标系中绘制这个棱锥，仅仅忽略掉每个点的z坐标。

（3）你看见了什么？如果按照上面说明正确地做了，你将会看见一个正方形，以及从角到角的两条对角线。它对应的是什么？它是当你从上往下看棱锥，而且当投影平面与棱锥的底平行时所得到的视图。

（4）现在重复这个练习，但是这一次忽略掉y坐标。这次你又看见了什么？请对此给出它的几何解释。

你所使用的这种投影类型是*正交平行投影*。在这种投影中，视图的方向总是与某个主轴平行，投影平面与观察方向正好成直角，而且视点位于无限远处。

这种投影给人一种“不真实”的印象，因为它们与我们日常的观察结果是不相吻合的。

195 我们知道如果沿着一条很长的路向远处看去，离我们视点远的一端道路的两边将会汇聚到一个点上（在无限远处）。平行投影不会形成这样的效果——在平行投影中，场景中的平行直线在投影图像中保持平行。（当然道路确实到处都一样宽，因此平行投影从这个角度保持了真实的一面——这就是为什么对于工程图应用平行投影更重要，而对于目标是产生真实感图像的计算机图形学不适合。）

透视投影

为了要得到现实世界中的视觉效果，我们需要透视投影。它所带来的效果更像自然视觉

效果。简单的透视投影如图9-3所示。这里视平面是 XY 平面（X轴看成是从书页指向外侧，遵从左手法则）。X、Y 和Z 轴可以看成第7章中的 UVN 系统。

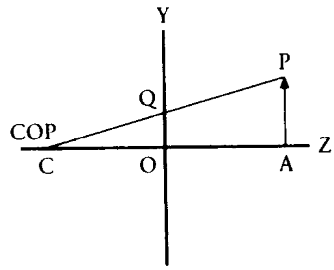


图9-3 透视投影

投影中心（COP）位于Z轴上，在视平面后面与视平面距离为d=OC处。点P(x，y，z)是3D中的点，它将被投影到XY平面上的Q点。三角形COQ和三角形CAP是相似三角形。因此有，

$$\frac{OQ}{OC}=\frac{AP}{AC} \tag{9-1}$$

如果Q点为(x'，y'，z')，则用坐标表示时有：

$$x'=\frac{xd}{d+z} \qquad y'=\frac{yd}{d+z} \tag{9-2}$$ 196

现在假设有一组参数定义了一个特定照相机（VRP、VPN和 VUV），由式（7-11）所定义的矩阵M可以构造出来。在WC中任何多边形可以用视图坐标重新描述，也就是说，使用矩阵M转变为图9-3相应的坐标系中。那么使用式（9-2）可以将多边形投影到视平面上。最后，在第5章中所描述的方法可以用来将多边形顶点映射到显示空间中。

规范框架

规范框架是一种基本布置，用来捕获透视投影和平行投影的基本要素。规范透视框架如图9-4所示。COP 点在点（0，0，－1）处，视平面与UV平面重合，而且视平面窗口在U和V两个轴向上都是在－1到 +1之间。因此视景体是个顶端位于COP点规则的棱锥（理论上在两个方向上可以扩展到无限远处）。

对于平行投影，其规范框架如图9-5所示，规范框架构成一个正交平行投影，投影方向为（0，0，－1），视平面窗口与透视投影情况相同。对于平行投影来说，视景体是个无穷的平行六面体，由U=±1和 V=±1 平面所构成。

设 p=(x，y，z)是VC坐标系中一个点，设p'是其在视平面上相应的投影点。对于正交平行投影，显然有 197

$$p'=(x,y,0) \tag{9-3}$$

从图9-4中我们可以看出，由相似三角形可以推得透视投影坐标如下：

$$p' = \left(\frac{x}{z+1}, \frac{y}{z+1}, 0\right) \tag{9-4}$$

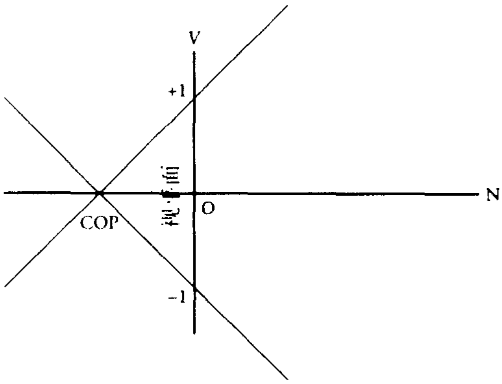


图9-4　透视投影的规范框架

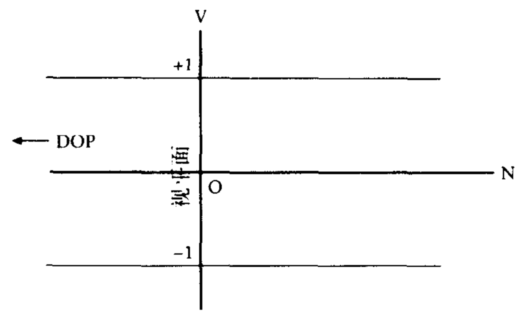


图9-5　平行投影的规范框架

转换到规范透视框架

透视投影一般的处理过程是，首先通过构造一个变换矩阵将场景转换到规范透视框架中，然后进一步由另外一个转换矩阵再将其转换到规范平行框架中。在这一上下文中，这个最后的空间被称做投影空间，它对于渲染管道在很多方面都有用处。

首先让我们考虑将图9-6所示的空间转换到图9-4所示的规范空间。设COP点坐标为（c_x，c_y，c_z）。设d是VP距离，（U_1，U_2，V_1，V_2）为VP窗口。构造一个矩阵P把一般的VC坐标系转变成规范坐标系有许多步骤需要执行。

（1）平移VP使之与UV平面重合。这需要应用如下矩阵：

198

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -d & 1 \end{bmatrix} \tag{9-5}$$

这里d是视平面距离。

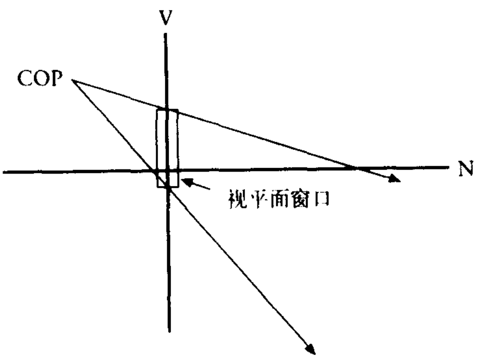


图9-6　观察坐标框架中的COP

（2）现在将COP从其新位置平移到N轴上。因此变换矩阵为

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -c_x & -c_y & 0 & 1 \end{bmatrix} \tag{9-6}$$

结果如图9-7所示。

注意到

$$\begin{aligned} D &= d - c_z \\ x_i &= U_i - c_x \\ y_i &= V_i - c_y \end{aligned} \tag{9-7}$$

这里 i=1，2。

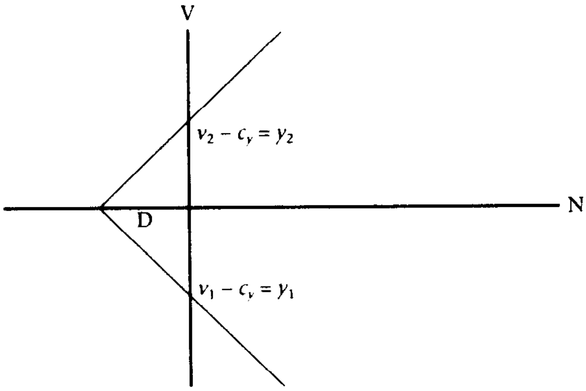


图9-7 转向规范框架

（3）改变视景体为一般的棱锥，裁剪平面将是$x=\pm(z+D)$和$y=\pm(z+D)$。容易证明通过下列矩阵可以达到这一点（参看附录9.1）：

$$\begin{bmatrix} \frac{2D}{\mathrm{d}x} & 0 & 0 & 0 \\ 0 & \frac{2D}{\mathrm{d}y} & 0 & 0 \\ -\frac{\mathrm{p}x}{\mathrm{d}x} & -\frac{\mathrm{p}y}{\mathrm{d}y} & 1 & 0 \\ -\left(\frac{\mathrm{p}x}{\mathrm{d}x}D\right) & -\left(\frac{\mathrm{p}y}{\mathrm{d}y}D\right) & 0 & 1 \end{bmatrix} \tag{9-8}$$

这里

$$\begin{aligned} \mathrm{d}x &= x_2 - x_1 \\ \mathrm{d}y &= y_2 - y_1 \\ \mathrm{p}x &= x_2 + x_1 \\ \mathrm{p}y &= y_2 + y_1 \end{aligned} \tag{9-9}$$ 199

（4）通过下列矩阵用伸缩因子$1/D$分别乘以X、Y 和 Z：

$$\begin{bmatrix} \frac{1}{D} & 0 & 0 & 0 \\ 0 & \frac{1}{D} & 0 & 0 \\ 0 & 0 & \frac{1}{D} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9-10}$$

现在执行矩阵乘法运算，依次将式（9-5）、式（9-6）、式（9-8）和式（9-10）中矩阵相乘得到最后的矩阵如下：

$$Q=\begin{bmatrix} \frac{2}{\mathrm{d}x} & 0 & 0 & 0 \\ 0 & \frac{2}{\mathrm{d}y} & 0 & 0 \\ -\left(\frac{\mathrm{p}x}{\mathrm{d}x}\right)\left(\frac{1}{D}\right) & -\left(\frac{\mathrm{p}y}{\mathrm{d}y}\right)\left(\frac{1}{D}\right) & \frac{1}{D} & 0 \\ -c_x\left(\frac{2}{\mathrm{d}x}\right)+\left(\frac{1}{D}\right)\left(\frac{\mathrm{p}x}{\mathrm{d}x}\right)c_z & -c_y\left(\frac{2}{\mathrm{d}y}\right)+\left(\frac{1}{D}\right)\left(\frac{\mathrm{p}y}{\mathrm{d}y}\right)c_z & -\left(\frac{\mathrm{d}}{D}\right) & 1 \end{bmatrix} \tag{9-11}$$

式（9-11）中所示矩阵将会把图9-4中所示的一般VC坐标系转换到图9-6所示的规范坐标系中。

规范投影空间

3D观察管道的变换部分中最后一个步骤是转换到图9-5中所示的规范投影空间。很容易看出可通过下列矩阵完成：

$$P=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9-12}$$

由这个矩阵转换之后所获得的空间称为投影空间（PS）。

现在假设在规范VC坐标系中有任意一个齐次点（x，y，z，1），其在 PS 中的等价点是（x，y，z，z+1）（在经过乘以矩阵P后）。因此等价的欧氏3D空间点通过对 X、Y 和 Z 分别乘以W得到，即：

$$\left(\frac{x}{z+1},\frac{y}{z+1},\frac{z}{z+1}\right)$$

因为这是PS中的一个点，将这个点投影到XY平面上显然只需忽略掉 Z坐标即可：

200

$$\left(\frac{x}{z+1},\frac{y}{z+1}\right)$$

通过与式（9-4）比较，我们知道这是最初点（x，y，z）对应的透视投影。

这个到PS的最后变换因此隐式地执行了透视投影。最为重要的一点在于它简化了可见性问题（隐藏面的删除），该问题我们还要在第13章中讨论。

合并前裁剪平面和后裁剪平面

在最后矩阵的导出中，没有考虑前裁剪平面和后裁剪平面。那么对于它们会有什么影响呢？如何将它们整合在一起，使得在最后的投影空间中前裁剪平面在0处，后裁剪平面在1处？

方法如下：假设在应用式（9-12）之前，前裁剪平面和后裁剪平面已经变成*Dmin*和*Dmax*。那么考虑变换：

$$
\begin{aligned}
z' &= \left(\frac{Dmax+1}{Dmax-Dmin}\right)\left(\frac{z-Dmin}{z+1}\right)\\
y' &= \frac{y}{z+1}\\
x' &= \frac{x}{z+1}
\end{aligned}
\tag{9-13}
$$

它完成透视变换。而且，当$z=Dmin$时，$z'=0$；当$z=Dmax$时，$z'=1$。用矩阵形式表示为：

$$
\begin{bmatrix}
1 & 0 & 0 & 0\\
0 & 1 & 0 & 0\\
0 & 0 & \dfrac{Dmax+1}{Dmax-Dmin} & 1\\
0 & 0 & \dfrac{Dmin\cdot(Dmax+1)}{Dmax-Dmin} & 1
\end{bmatrix}
\tag{9-14}
$$

现在用这个矩阵代替式（9-12），则最后空间将会保证前裁剪平面在0处而后裁剪平面在1处。

变换平面距离

我们知道COP点是(c_x, c_y, c_z)。设d是视平面距离，*dmin*和*dmax*分别是前裁剪平面和后裁剪平面的距离。在式（9-11）中的变换矩阵Q将VC空间转换到如图9-4中所示的规范观察空间。 201

让我们看一下在这些变换下VPD和d发生了什么变化。为了要通过矩阵转换VPD（只是一个距离），我们需要把它表示成点的形式。我们将它表示成形式为（_、_、d、1）的任何点，这里使用“_”代表“不关心”它的值是多少，因为对于VPD，我们只关心它从原点出发在z方向上的距离。现在应用式（9-5）中矩阵，得到点（_、_、0、1）。式（9-6）、式（9-8）和式（9-10）中的矩阵对z坐标不再产生影响，所以在规范VCS中，VPD已经变成0，这是我们所需要的结果。

类似地，我们可以观察在这些变换下FCP和BCP发生了什么变化。FCP可以表示为（_、_、*dmin*、1）。使用式（9-5）我们得到（_、_、$dmin-d$、1），式（9-6）和式（9-8）的矩阵对z坐标没有影响，式（9-10）的伸缩因子为D。因此在规范VCS中前裁剪平面和后裁剪平面被转换为：

$$
\begin{aligned}
Dmin &= \frac{dmin-d}{D}\\
Dmax &= \frac{dmax-d}{D}
\end{aligned}
\tag{9-15}
$$

从式（9-7）可以看出这里$D=d-c_z$。

9.4　合成矩阵

现在让我们从一个WC点开始，它必须被转换到规范VC坐标中，首先将它转换到VC，然后再转换到规范VC上。为了要完成这一项任务，首先乘以矩阵M然后再乘以矩阵Q。所以，给定WC中的一个点 $p=(x, y, z)$，所得到的在规范VC中的点将是$(x, y, z, 1)MQ$。这里我们不去计算两矩阵乘积，而是使用矩阵M和Q的乘积矩阵。设为T，即$T=MQ$，如下所示：

202

$$T=\begin{bmatrix} \frac{2u_1D-n_1\mathrm{p}x}{D\mathrm{d}x} & \frac{2v_1D-n_1\mathrm{p}y}{D\mathrm{d}y} & \frac{n_1}{D} & 0 \\ \frac{2u_2D-n_2\mathrm{p}x}{D\mathrm{d}x} & \frac{2v_2D-n_2\mathrm{p}y}{D\mathrm{d}y} & \frac{n_2}{D} & 0 \\ \frac{2u_3D-n_3\mathrm{p}x}{D\mathrm{d}x} & \frac{2v_3D-n_3\mathrm{p}y}{D\mathrm{d}y} & \frac{n_3}{D} & 0 \\ -\left(\frac{2(qu)D-(qn)(\mathrm{p}x)+2c_xD-(\mathrm{p}x)c_z}{D\mathrm{d}x}\right) & -\left(\frac{2(qv)D-(qn)(\mathrm{p}y)+2c_yD-(\mathrm{p}y)c_z}{D\mathrm{d}y}\right) & -\left(\frac{qn+d}{D}\right) & 1 \end{bmatrix} \tag{9-16}$$

这里下列各项已经事先被定义。

U_1, U_2, V_1, V_2 确定了VP窗口，且

$$\begin{aligned} x_i &= U_i - c_x \ (i=1,2) \\ y_i &= V_i - c_y \ (i=1,2) \\ \mathrm{d}x &= x_2 - x_1 = U_2 - U_1 \\ \mathrm{d}y &= y_2 - y_1 = V_2 - V_1 \\ \mathrm{p}x &= x_1 + x_2 = U_1 + U_2 - 2c_x \\ \mathrm{p}y &= y_1 + y_2 = V_1 + V_2 - 2c_y \end{aligned} \tag{9-17}$$

(q_1, q_2, q_3)是VRP且

$$\begin{aligned} qu &= \sum_{i=1}^{3} q_i u_i \\ qv &= \sum_{i=1}^{3} q_i v_i \\ qn &= \sum_{i=1}^{3} q_i n_i \end{aligned} \tag{9-18}$$

是VRP与矢量u、v和n的内积，矢量u、v和n由式（7-1）、式（7-2）和式（7-3）定义。

最后结果是，假设WC中的点为(x, y, z)，在规范VC中的相应点为$(x, y, z, 1)T$，这里T由式（9-16）给出。

9.5　计算视图矩阵T

当我们用C语言实现先前小节的结果时，最好把照相机做为一个抽象数据类型（或类）。`Camera`应该包含定义矩阵T所需要的所有信息，如下所示：

```
/*we distinguish between points and vectors for semantic reasons*/

typedef double Matrix[4][4];

typedef struct{
  double x,y,z;
} Vector;

typedef struct{
  double x,y,z;
} Point3D;
/*we assume a number of functions:*/

double normVector3D(Vector3D *v);
/*returns the norm of the Vector3D *v*/

double dotProductVector3D(Vector3D *v1, Vector3D *v2);
/*returns the dot project (*v1).(*v2)*/

void crossProductVector3D(Vector3D *v1, Vector3D *v2, Vector3D *vout);
/*returns the cross project (*v1)*(*v2) and puts the result in vout*/

void differencePoint3D(Point3D *p1, Point3D *p2, Vector3D *vdiff);
/*returns the vector that is the difference of two points p1-p2*/

void normalizeVector3D(Vector3D *v, Vector3D *vout);
/*normalizes *v to have length 1, with result returned in vout*/

void transformPoint3D(Point3D *p, Matrix m);
/*does the matrix multiplication (p->x,p->y,p->z,1)*m
to return a new point in p*/

typedef struct{
  /*public:*/
  DRAWABLE drawable;           /*a drawing surface e.g. an X Window*/
  Point3D vrp;                 /*view reference point*/
  Vector3D vpn;                /*view plane normal*/
  Vector3D vuv;                /*view up vector*/
  Point3D cop;                 /*centre of projection*/
  double vpd,fcp,bcp;          /*view plane distance*/
                               /*front and back clipping planes*/
  double u1,u2,v1,v2;          /*view plane window*/
  int width, height;           /*of drawable*/

  /*private:*/
  double aX,bX,aY,bY;          /*converts from 2D window -> display*/
  Point3D wc_cop;              /*COP expressed in WC*/
  double Dmin,Dmax,dr;
  Matrix T;
} Camera;
```

203

现在让我们看一下一些函数的实现：

```
void setVRP(Camera *camera, double x, double y, double z)
/*sets the View Reference Point*/
{
  camera->vrp.x = x;
  camera->vrp.y = y;
  camera->vrp.z = z;
}
```

204

```
void setVPN(Camera *camera, double x, double y, double z)
/*sets the View Plane Normal*/
{
  camera->vpn.x = x;
  camera->vpn.y = y;
  camera->vpn.z = z;
}

void setVPWindow(Camera *camera, double xmin, double xmax, double
ymin, double ymax)
/*sets the View Plane Window*/
{
  camera->U1 = xmin;
  camera->U2 = xmax;
  camera->V1 = ymin;
  camera->V2 = ymax;
}
```

对于其他函数也同样，应用程序所设的每组参数都会有一个对应函数，其职责是在数据结构中储存适当的值。这些函数当然也应该有误差检查，举例来说，检查VPN矢量与VUV矢量所形成的角度是否为0或者180度。现在我们需要另外一个函数，假设在结构中储存有已知信息计算矩阵*T*。依据模型，我们可以肯定这个函数会将照相机参数设置为那些指定值，也就是使得照相机在这些固定参数组间切换。这个函数的实现现在全部给出。

```
void click(Camera *camera)
/*computes viewing matrix based on existing parameter settings*/
{
  Vector3D u,v,n,vout;
  double px,py,dx,dy,qu,qv,qn,D,Ddx,Ddy;
  int i;
  double U[3],V[3],N[3],c[3],cx,cy,cz;

/*(a) create the M matrix*/
  /*(EQ 212) to (EQ 214)*/
  /*normalize VPN and put into n*/
  normalizeVector3D(&camera->vpn,&n);

  /* vout = n x VUV */
  crossProductVector3D(&n,&camera->vuv,&vout);
  /*computes u = (n x VUV)/|n x VUV| */
  normalizeVector3D(&vout,&u);
  /*computes v = u x n */
  crossProductVector3D(&u,&n,&v);

  /*for last row of M matrix (have to cast point to vector)*/
  /*(EQ 221), (EQ 222) and (EQ 9.18)*/
  qu = dotProductVector3D((Vector3D *)&camera->vrp,&u);
  qv = dotProductVector3D((Vector3D *)&camera->vrp,&v);
```

205

```
  qn = dotProductVector3D((Vector3D *)&camera->vrp,&n);
 /*(b) compute the T matrix*/
   /*(EQ 9.16)*/
   dx = camera->U2 - camera->U1;
   dy = camera->V2 - camera->V1;
   px = camera->U1 + camera->U2 - 2.0*camera->cop.x;
   py = camera->V1 + camera->V2 - 2.0*camera->cop.y;

   /*(EQ 9.15)*/
   D = camera->vpd - camera->cop.z;
   camera->Dmin = (camera->fcp - camera->vpd)/D;
```

```
    camera->Dmax = (camera->bcp - camera->vpd)/D;
    Ddx = D*dx;
    Ddy = D*dy;

    /*need for transformation to Projection Space. See "Incorporating
    the Front and Back Clipping Planes" on page 000.*/
    camera->dr = (camera->Dmax + 1.0)/(camera->Dmax - camera->Dmin);

    /*now we have all the ingredients for computing T:)*/
    /*first do the first three rows and columns*/
    /*convert u,v,n to arrays -- for convenience*/
    U[0] = u.x; U[1] = u.y; U[2] = u.z;
    V[0] = v.x; V[1] = v.y; V[2] = v.z;
    N[0] = n.x; N[1] = n.y; N[2] = n.z;

    for(i = 0; i<3; ++i){
      camera->T[i][0] = (2*U[i]*D - N[i]*px)/Ddx;
      camera->T[i][1] = (2*V[i]*D - N[i]*py)/Ddy;
      camera->T[i][2] = N[i]/D;
      camera->T[i][3] = 0.0;
    }

    /*now do the last row*/
    camera->T[3][0] =
      -(2*qu*D-qn*px+2*camera->cop.x*D-px*camera->cop.z)/Ddx;
    camera->T[3][1] =
      -(2*qv*D-qn*py+2*camera->cop.y*D-py*camera->cop.z)/Ddy;
    camera->T[3][2] = -(qn + camera->vpd)/D;
    camera->T[3][3] = 1.0;

  /*(c) compute the WC cop*/
    /*compute the COP expressed in WC = cop*R'+vrp,
      where R = rotation part of M
      R' is the transpose of R, ie, has rows: [u1,u2,u3]
      [v1,v2,v3][n1,n2,n3]
    */
    for(i=0;i<3;++i)  c[i] = camera->cop.x*U[i]+camera->
    cop.y*V[i]+camera->cop.z*N[i];
    camera->wc_cop.x = c[0] + camera->vrp.x;
    camera->wc_cop.y = c[1] + camera->vrp.y;
    camera->wc_cop.z = c[2] + camera->vrp.z;

  /*(d) do the 2D window->display transformation*/
    /*set up the 2D window->display transformation*/
   /*now the view plane window will have been transformed to
   -1 <= x <= +1, -1 <= y <= +1 */

   camera->bX = (double)camera->width/2.0;
   camera->aX = camera->bX;

   camera->bY = (double)camera->height/2.0;
   camera->aY = camera->bY;

   /*adjust y to turn the y axis right way up*/
   camera->aY = camera->height - 1 - camera->aY;
   camera->bY = -camera->bY;
}
```

206

函数click立即计算从WC到投影空间转换所需的每一部分数据，包括最后的透视投影，即渲染一个WC多边形的信息。这个函数包含四个主要部分：

创建矩阵*M*。它使用VRP、VPN和VUV通过式（7-1）到式（7-11）计算矩阵*M*。

计算矩阵*T*。即计算式（9-16）中整个矩阵*T*。

计算WC中的COP。COP是在VC中定义的。然而，我们稍后所需的一个操作是去确定是否COP点位于任意多边形相应平面的"前侧"或"后侧"（第8章）。这被用于背面删除——对任意正确定义了的多面体（满足欧拉方程，并且对于其"外侧面"来说，其多边形的顶点是按反时针的顺序排列的），一个背向COP的多边形是不能从COP看见的，因为它至少被多面体的一个其他面所遮挡。假设其平面方程是：

$$l(x, y, z) = ax + by + cz - d = 0 \tag{9-19}$$

那么如果（*X*、*Y*、*Z*）是任意点，如果（*X*、*Y*、*Z*）在平面的"正半空间"中，则$l(X, Y, Z) > 0$。因此，为了确定是否COP在外侧，我们需要将COP代入式（9-19）中。

然而，COP是在VC中表示的而非在WC中，式（9-19）却需要一个WC点（因为平面方程是用WC表示的）。因此必须从VC转换到WC。这实际上是非常容易的。我们知道矩阵*M*将从WC转换到VC，因此我们需要的是*M*的逆矩阵。

从式（7-4）和式（7-10）中，我们有：

207

$$M = \begin{bmatrix} R & 0 \\ -qR & 1 \end{bmatrix} \tag{9-20}$$

这里*q*是VRP。

同时，我们通过构造知道*R*是一个正交旋转矩阵（即它的逆矩阵与其转置矩阵R^T一致）。因此很容易得出*M*的逆矩阵为：

$$M^{-1} = \begin{bmatrix} R^T & 0 \\ q & 1 \end{bmatrix} \tag{9-21}$$

因此用WC表达的COP位置是：

$$(cop, 1)M^{-1} = cop \cdot R^T + vrp \tag{9-22}$$

从几何角度看，如果我们把COP视为在VC中的一个矢量，那么R^T矩阵将它转换成在WC中对等的矢量。通过VRP转换它以便获得必需的WC表示。

从二维窗口到显示平面的变换。这利用了第5章中的材料。应用投影变换后，视平面窗口将变成$-1 < x < 1$，$-1 < y < 1$。经过裁剪（下一章中介绍）之后，所有点将处于这个窗口的范围之内。我们假设显示区域的尺寸为（用像素表示）$width \times height$。利用先前的结果，我们将显示点（视口）(x', y')与窗口点(x, y)之间的关系表示为如下：

$$\begin{aligned} x' &= Vxmin + \left(\frac{dVx}{dWx}\right)(x - Wxmin) \\ y' &= Vymin + \left(\frac{dVy}{dWy}\right)(y - Wymin) \end{aligned} \tag{9-23}$$

此时我们有$Wxmin = -1.0$、$Wxmax = 1.0$，$dWx = 2.0$，对于*y*有相似结论。而且有$dVx = width$和$dWx = height$，化简方程为：

$$\begin{aligned} x' &= \left(\frac{width}{2}\right)(x + 1) \\ y' &= \left(\frac{height}{2}\right)(y + 1) \end{aligned} \tag{9-24}$$

x'和y'应该被限定为整数。

最后，许多显示系统将Y轴定义为从上端至下端逐渐增加，所以 y=0 在显示器的顶端。因此我们需要颠倒 y 的方向，通过由最大的像素值（*height* －1）减y值得到。

9.6 技术整合

实际上，我们在WC中有一个多边形，例如p_i=（x_i，y_i，z_i）（i=0，…，n－1），$p_n \equiv p_0$。我们需要转换多边形到规范VC坐标系中，那么我们就裁剪它，然后转换结果到规范投影空间（PS）（即到方盒形空间中）。 208

为了做最后的一步，对每个点使用矩阵view->T将它转换到规范 VC 中。（实际上，我们会使用点的一个拷贝，并非重写在场景数据结构中储存的值。）

如果我们对多边形中每个点都执行了一遍，它的顶点将会表达成规范 VC的形式。我们可以接着通过各个裁剪平面对它进行裁剪：

$$X=\pm(Z+1)\text{和 } Y=\pm(Z+1),\ Z=Dmin \text{ 和} Z=Dmax$$

裁剪过程执行后我们得到了一个新的多边形（或者根本就什么也没有留下），用规范 VC 顶点q_0，q_1，…，q_{m-1}表示。现在它们每个都要被转换到规范 PS中，通过使用上面结果。举例来说，点q转换为PS 中的点，假设这个点为r点，其转换如下所示：

```
zplus1 = q.z + 1.0;
r.x = q.x/zplus1 ;
r.y = q.y/zplus1 ;
r.z = view->DR*((q.z - view->Dmin)/zplus1);
```

最后的点r=（r.x，r.y，r.z）在PS 中，所以其投影点是（r.x，r.y）。使用式（9-23），这变成了在显示坐标系中的点坐标（round (ax+bX*r.x)，round (aY+bY*r.y)），对应的z深度值是 r.z。我们使用 z 深度来解决隐藏表面问题。

9.7 视图实现与场景图的结合

实际上，场景储存为一个对象层次结构，每个对象用它自己的局部坐标系来表示。从第 8 章中我们知道，CTM（当前变换矩阵）是与层次结构中每个对象相关联的，这个矩阵的作用是从局部坐标系转换到全局坐标系（世界坐标）。一种做法是，首先通过乘以对象的 CTM 对其所有顶点进行变换，然后再重新对这个对象的所有顶点乘以矩阵T。另一种更有效率的做法是一个对象一个对象地将CTM和T结合在一起。这可以通过下面的函数来说明，它显示相对于照相机的一个对象节点。

```
void displayObject(Camera *camera, GObject *object)
/*This displays all the faces associated with the object of the  camera*/
{
  FaceArray *farray;
  register int i;
  Matrix T,M;
  Point3D p;

  farray = object->farray;
/*update the camera transformation matrix by the CTM of this object*/
copyMatrix(camera->T,T);
multiplyMatrix(object->CTM,camera->T,M);
copyMatrix(M,camera->T);
```

209

```
    /*transform the wc_cop into the local coord system -
      for back face elimination*/
    p = camera->wc_cop;
    transformPoint3D(&camera->wc_cop,object->invCTM);

    for(i=object->startFace; i < object->startFace+object->numFaces; ++i)
      displayFace(camera,atFaceArray(farray,i));

    /*restore camera transformation matrix*/
    copyMatrix(T,camera->T);

    /*restore wc_cop*/
    camera->wc_cop = p;
}
```

这个函数执行下列各项运算:

(1) 保存当前照相机变换矩阵 (camera->T) 到局部矩阵T。

(2) 用对象的 CTM 与camera->T的乘积重写camera->T，形成一个新的矩阵，由它来负责完成从局部坐标到规范观察坐标的转换。

(3) 将由 WC 表示的 COP 转换到对象的局部坐标空间中 (通过乘以逆CTM矩阵——这也是为了背面删除)。

(4) 对对象的表面执行一般的渲染过程。现在这个对象已经得到渲染，存储照相机和COP 的先前状态。

9.8 在OpenGL中观察

在第8章中我们讨论了 OpenGL最为基本的结构及操作，未曾实施任何特殊的技术策略。(这当然没有超出由基本假设所设定的范围，所谓的基本假设比如场景最终都将用多边形来显
210 示。) 在观察的上下文中，OpenGL 遵循相同的思想：它提供一个矩阵 (堆栈)，即模型视图矩阵，将对象从对象的局部坐标转换到观察坐标 (眼睛坐标)，另外还有一个投影矩阵。在第8章中我们介绍了模型视图矩阵的“模型”部分，这里我们将分析它的“视图”部分以及投影矩阵。

首先，让我们分析一下将如何使用OpenGL设施来实现在这一章中前面所讨论的照相机模型。首先考虑转换到VC系统的矩阵计算。这在clickView_GL函数中给出。

```
typedef struct{
  Matrix m;
} RotationMatrix;

void clickView_GL(Camera_GL *camera)
/*does the transformation to viewing coordinates: using VRP, VPN, VUV.
Transforms to a RHS looking down negative z-axis*/
{
  Vector3D u,v,n,vout;
  GLfloat m[16];
  RotationMatrix r;

  /*normalize VPN and put into n*/
  normalizeVector3D(&camera->vpn,&n);

  /* vout = VUV x n */
  crossProductVector3D(&n,&camera->vuv,&vout);
  normaliseVector3D(&vout,&u); /*computes u = (VUV x n)/|VUV x n| */
```

```
crossProductVector3D(&u,&n,&v); /*computes v = n x u */

/*construct the viewing matrix, and also store it*/

r.m[0][0]=m[0]=u.x; r.m[0][1]=m[1]=v.x; r.m[0][2]=-(m[2]=(-n.x));
m[3] = 0.0;
r.m[1][0]=m[4]=u.y; r.m[1][1]=m[5]=v.y; r.m[1][2]=-(m[6]=(-n.y));
m[7] = 0.0;
r.m[2][0]=m[8]=u.z; r.m[2][1]=m[9]=v.z; r.m[2][2]=-(m[10]=(-n.z));
m[11] = 0.0;
m[12] = 0.0; m[13] = 0.0; m[14] = 0.0; m[15] = 1.0;

camera->R = r;

/*the OpenGL calls*/
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glLoadMatrixd(m);
glTranslated((GLdouble)(-camera->vrp.x),
   (GLdouble)(-camera->vrp.y),
   (GLdouble)(-camera->vrp.z));
}
```

这个子程序的第一部分与先前的相同，是计算 n、u和 v 三个矢量。用它们来构造 M 矩阵。储存了两个表示，一个存在于RotationMatrix的内部，另一个只在OpenGL中使用（m）。 211
有一点很重要，那就是 OpenGL 假设始终使用右手坐标系，所以在矩阵m的第三列中负值是需要的。

另外一点也是很重要的，那就是实际上OpenGL所定义的点都是列向量，因此使用矩阵与点相乘（在本书中我们使用它的补充约定）。在 OpenGL 中要使用本书中矩阵的转置矩阵。然而，注意到矩阵是一个包含16个双精度实数的单维数组的形式，矩阵m在OpenGL中的表示如下所示：

$$\begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix} \tag{9-25}$$

构造完M矩阵，就由glMatrixMode（GL_MODELVIEW）来设置当前矩阵堆栈为模型视图堆栈，它的初值为单位矩阵。然后，glLoadMatrix（m）将当前模型视图矩阵乘以m，因为当前矩阵是单位矩阵，所以这就得到了所需的结果——事实上它几乎就是式（7-7）中的矩阵。然而需要注意的是，我们需要考虑 VRP（见式（7-10））。glTranslated执行前模型视图矩阵与相应的平移矩阵相乘，获得所需要的矩阵（见式（7-11））。

下一步我们需要构造矩阵Q的等价形式（见式（9-11））——投影矩阵。OpenGL定义了一个函数：

```
glFrustum(GLdouble left, GLdouble right,
          GLdouble bottom, GLdouble top,
          GLdouble near, GLdouble far)
```

它在近裁剪平面上为视平面窗口确定了一个投影矩阵，该视平面窗口的两个顶角分别为（left，bottom，－near）和（right，top，－near）。这两个角分别映射到观察窗口的左下角和

右上角。投影中心假定为（0，0，0）。注意这是假设“场景”位于负Z轴上（事实上这与使用左手坐标系作为观察坐标系完全相同，只是需要对Z轴做一个负向标记）。

在现在的模型中，我们将视平面窗口定位在视平面上（在距离camera->vpd处），而不是在前裁剪平面上。同时，COP无需是原点，可以在任意点（cx，cy，cz）上。因此为了使用OpenGL函数，我们必须求出视平面窗口在前裁剪平面上的投影，并将COP脱离原点位置。

图9-8说明了如何计算在前裁剪平面上一个适当的VP窗口。考虑有一个VP窗口y坐标在VP上距原点的距离为d。我们的目标是要求出在前裁剪平面（FCP）上的对应坐标(y')，假设前裁剪平面距原点的距离为f。由相似三角形，我们有：

212

$$\frac{y}{d}=\frac{y'}{f} \tag{9-26}$$

$$y'=\left(\frac{f}{d}\right)y$$

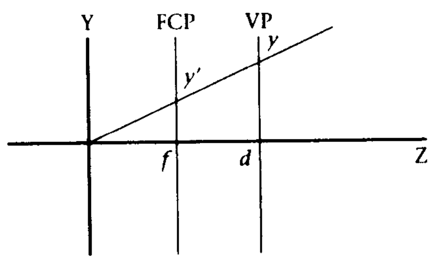


图9-8 计算前裁剪平面上的VPW

所有这些都整合在函数clickProject_GL中了。注意当前矩阵（堆叠）是如何设置成投影矩阵的，然后设置成单位矩阵，因为由glFrustum所确定的矩阵将乘以当前矩阵。

```
void clickProject_GL(Camera_GL *camera)
/*sets up the projection transformation*/
{
  GLdouble fcp,bcp,a,cx,cy,cz,umin,umax,vmin,vmax;

  /*compute the VPW given the COP at the origin, and assume that
  the VPW has been specified on the VPD*/
  cx = camera->cop.x; cy = camera->cop.y; cz = camera->cop.z;
  fcp = (camera->fcp - cz);
  bcp = (camera->bcp - cz);
  a = fcp/(camera->vpd - cz);
  umin = a*(camera->U1 - cx);
  umax = a*(camera->U2 - cx);
  vmin = a*(camera->V1 - cy);
  vmax = a*(camera->V2 - cy);

  glMatrixMode (GL_PROJECTION);        /* prepare for and then */
  glLoadIdentity ();                   /* define the projection */
  glFrustum(umin,umax,vmin,vmax,fcp,bcp);
  glTranslated(-cx,-cy,cz);
}
```

构造完这些矩阵，下个问题是该如何在渲染中使用它们。我们考虑两个问题，第一是对单个面的渲染，第二个是显示对象层次结构中的一个对象，这里对象是表示在局部坐标系中的。

```
void displayFace_GL(Face *face)
/*This displays the (convex) polygon associated with the face*/
{
   int n,i;
   VertexArray *va;
   Index *index;
   Point3D p;
   PlaneEq plane;
   /*determine material properties according to rules above*/          213
   determineMaterial(face);

   /*get the vertex array and index to first point*/
   va = vertexArrayOfFace(face);
   index = face->first;

   glBegin(GL_POLYGON);
     plane = *planeEqOfFace(face);
     glNormal3d(plane.a,plane.b,plane.c);

     n = numVerticesInFace(face);
     for(i=0; i<n; ++i){
       p = *atVertexArray(va,valueAtIndex(index));
       glVertex3d(p.x,p.y,p.z);
       index = nextIndex(index);
     }
   glEnd();
}
```

显示一个面是简单的。面的坐标可以从面的数据结构中抽取出来，在glBegin (GL_POLYGON) 和glEnd () 之间的程序段中给出。从光照角度考虑，面的法向需要设定，而且要确定材质属性。这在第6章中已经讨论过了。

```
void displayObject_GL(GObject *object)
/*This displays all the faces associated with the object*/
{
   FaceArray *farray;
   int i;
   Material *material;
   short disable;

   /*do nothing if this object has no geometry*/
   if(!HASGeometry(object)) return;

   /*determine the material*/
   if(material=object->material) {
     glMaterialfv(GL_FRONT,GL_AMBIENT,material->ambient);
     glMaterialfv(GL_FRONT,GL_DIFFUSE,material->diffuse);
     glMaterialfv(GL_FRONT,GL_SPECULAR,material->specular);
     glMaterialfv(GL_FRONT,GL_SHININESS,material->shininess);
     glShadeModel(material->model);
     disable=0;
     if(material->opacity==Transparant) {
       glEnable(GL_BLEND);
       disable=1;
     }
   }
   else setDefaultMaterial();
  /*get the object's face array*/                                       214
  farray = object->farray;
  /*make sure we're dealing with modelview matrix*/
```

```
    glMatrixMode(GL_MODELVIEW);

    /*pushes and duplicates current matrix*/
    glPushMatrix();

    /*multiply CurrentMatrix*CTM*/
    glMultMatrixd((GLdouble *)object->CTM); /*works because of double
    transpose*/

    for(i=object->startFace; i < object->startFace+object->numFaces; ++i){
      displayFace_GL(atFaceArray(farray,i));
    }

    /*restore camera transformation matrix*/
    glPopMatrix();

    if(disable) glDisable(GL_BLEND);
}
```

显示一个对象也是容易的（忽略掉与材质有关的部分）。首先，在对象层次结构中每个节点实际上并不是必须存储几何信息的（它可能仅仅是为变换保留位置），在这种情况下什么也不需要做。其次，在确定模型视图矩阵是当前矩阵后，我们就复制当前模型视图矩阵并将它压入模型视图矩阵堆栈的顶端（为今后恢复之用），因为每个对象有它自己的当前变换矩阵。接下来，我们将当前模型视图矩阵乘以对象的CTM，这个乘积将成为新的模型视图矩阵。当前模型视图矩阵应该仅仅对应于当前照相机设置的矩阵。这样每个面得到显示（记住每个面的坐标是局部的）。最后，最初的模型视图矩阵被恢复。

假设M是当前照相机矩阵，它将WC转换到VC。又假设C是这个对象的CTM。那么，

（1）M变成堆栈的顶端，当前模型视图矩阵还是M。

（2）当前模型视图矩阵变成CM。

215 （3）渲染面。

（4）将当前模型视图矩阵变成M。

```
void displayObjectTree_GL(GObject *object)
/*displays the entire subtree starting with object as root*/
{
    int i;

    /*do nothing if nothing there*/
    if(object == (GObject *)0) return;
    displayObject_GL(object);

    /*display the children*/
    if(object->n > 0){
      for(i=0; i<object->n; ++i) displayObjectTree_GL(*(object->child+i));
    }
}
```

渲染层次结构的整个一棵子树是简单的，只要递归调用函数displayObjectTree即可。如果对象是空的，则什么也无需做；否则就渲染这个对象，并递归地显示它的全部孩子（如果有的话）。

最后，让我们看一下由GLU实用程序库所提供的观察模型，其中包含了一定的“策略”——有两个函数用来定义到VC的变换和投影矩阵。

```
gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
          GLdouble centerx, GLdouble centery, GLdouble centerz,
          GLdouble upx, GLdouble upy, GLdouble upz)
```

点e=(eyex、eyey、eyez)可以看成WC的COP点，点c=(centerx、centery、centerz)可以看成是相应于场景中某个相关点的VRP。

因此VPN是 $n=c-e$。矢量 u=（upx、upy、upz）是视图的上方矢量。

这就产生了一个把n映射到负Z轴的矩阵（我们知道OpenGL是假设为RHS的），e代表原点，c代表显示视口的中心。

为了要确定投影矩阵，可以使用函数gluPerspective：

```
gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble znear,
GLdouble zfar)
```

它确定了在y方向（fovy的单位为角度）上视角的大小，确定了在前裁剪平面znear上VP窗口的宽度和高度的比例。后裁剪平面在zfar处。

在附录9.2中我们将继续第8章中“方盒堆栈”的例子。我们使用GLUT实用程序库来构造OpenGL窗口和定义回调函数。从主函数开始我们初始化GLUT、设定显示窗口的大小和显示模式（更多内容将在后面的各章中介绍），创建和映射窗口。所定义的回调函数确定了将会发生的事情：

- 显示需要随时刷新（glutDisplayFunc）。 216
- 窗口可以随时变化（glutReshapeFunc）。
- 什么事情也不发生（glutIdleFunc）。

在初始化中此刻要观察的惟一部分是使用gluLookAt来确定视图。

在回调函数reshape中调用了gluPerspective，因为窗口高度与宽度比在有窗口改变事件时会发生变化。

每当没有任何其他事件发生时，回调函数rotate被执行，它应用旋转矩阵，旋转所显示的对象。

display回调函数显示一组堆砌在一起的立方体，并交换渲染缓冲区以便保证动画无闪烁现象。程序的余下部分都在第8章中讨论过了。

9.9 创建3D立体视图

建立立体视图

在第1章中，我们介绍了3D场景的立体视图的概念，立体视图是通过分别构造左右眼图像，让视觉记忆融合这两帧图像以形成一幅具有立体感的3D图像。我们注意到在一幅图像中除了立体感以外还有许多其他的深度线索，例如线性透视、纹理梯度等等，但是立体感对近域对象是特别重要的。在这一小节中我们考虑如何能够通过照相机模型装置使得创建3D立体视图变得很容易。

图9-9给出了左右眼的一个示意性视图和投影场景的一个图像平面。IPD是双眼瞳孔间距离，即两眼之间的距离。点p_1是图像平面靠近眼睛一侧的一个点，从右眼睛看去它投影在图像平面上的R_1点，从左眼睛看去它投影在L_1点。同样地，p_2是在图像平面远离眼睛的一侧，它对于右眼和左眼来说分别投影在图像平面的R_2点和L_2点上。对于单一场景点的两个投影图像

点称为同源。举例来说，R_1和L_1是同源的。通常，当$R-L>0$时，它称为正水平视差（对应于p_2这种情况），而$R-L<0$则称为负水平视差。对应的术语同样可以用于垂直视差的定义。当左右眼睛图像融合，p_1和p_2将会成为3D中的一个虚拟点，第一个在立体视平面之前（负水平
217 视差），第二个在立体窗口的后面（正水平视差）。在这个投影下图像平面上的所有点都是不变的，也就是说，图像平面映射到它自身。

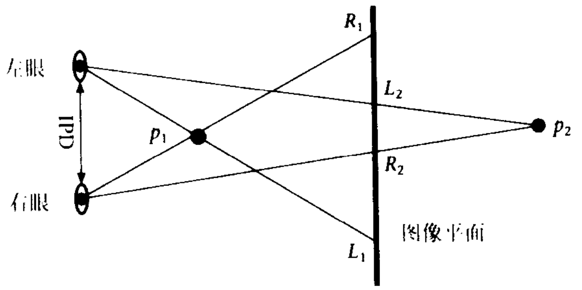


图9-9　立体对的水平视差

图9-10说明了立体重叠的思想。虽然有两个独立的图像，对每只眼睛各有一幅，它们所使用的是相同的一套观察设备——它们所使用的是相同的图像平面，具有相同的视平面法向、相同的视图上方矢量以及视平面窗口。只有投影中心是不同的。因为每个眼睛使用相同的视平面窗口，所以将会在它们各自的视景体上有一个重叠。这里所显示的情形是100%立体重叠，因为每个视图有完全相同的视平面窗口。让每个眼睛有独立的视平面窗口也是可能的，这样在两个窗口之间因而也就在两个投影之间有一个公共的重叠区域存在。这种重叠的百分比大小决定了最后的总视图在多大程度上是立体的，同时也决定了周边图像（非立体图像）的大小。若两眼之间间距IPD很宽，同时两幅图像重叠成分较小，则会影响虚拟图像的深度，而且会在图像融合方面带来比较大的困难和不舒适感。

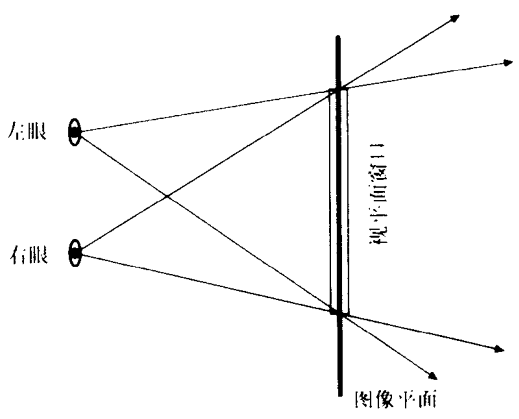


图9-10　立体重叠

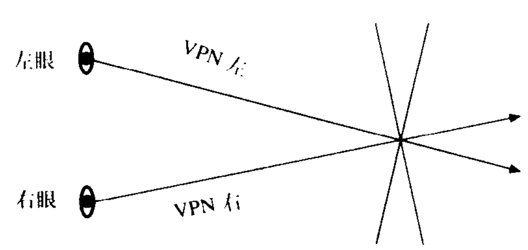


图9-11　通过旋转形成立体图像

对于左右眼睛视图只有投影中心不同，这一点是很有意思的。图9-11给出了另一种产生
218 立体图像的方法。这里左右眼睛位置可以看成是视图参考点，对应的COP点坐标都是（0, 0, 0）。现在每个都有其自己的视平面法向、视平面（视图上方矢量必须是相同的）。因此它们可以被认为是彼此的旋转。然而，不难证明（Hodges and McAllister，1993）这种方式所产生的是一

个非平坦的虚拟图像平面，而且也产生垂直视差。前者产生失真，两者都在融合图像方面引起不舒适感。

对立体视图计算方面的全面讨论可以参见Hodges和 McAllister 的著作（1993）。尤其是他们列出了许多应该考虑在内的有关建立理想立体视图的因素。其中一些列举如下：

- 利用左右两幅图像之间的一致性。这意味着两图像除了水平视差之外都是相同的。尤其是，在两幅图像中颜色和亮度对于同原点来说应该是一样的。
- 垂直视差应该为零，以便在融合图像方面避免不舒适感。
- 在深度和舒适性之间一个好的折衷是选择视平面位置，使得大约一半的视差值为正，一半为负。然而，通常情况是，观察者与显示器的距离越大，可以容忍的视差就会越大。

立体视图与照相机模型

彩图9-12给出了棱锥从顶端看时的一个视图。对棱锥的定义和基本的观察设定如图9-13所示。注意，VRP 与 XY 平面距离为100个单位，XY平面是棱锥的基座所在平面。图9-12这个单一视图的观察参数在表9-1中列出。

表9-1 图9-12中的观察参数

参 数	值
VRP	(5,5,100)
VPN	(0,0,−1)
VUV	(0,1,0)
COP	(0,0,0)
VPWindow	对X和Y，−10到10
VPDistance	80

现在我们将产生这个场景的三个立体视图，第一个是视平面位于棱锥之后，第二个是视平面位于棱锥中间，将棱锥平分为二。第三个是视平面位于棱锥之前。无论哪种情况，我们都设定瞳孔间距为3，这是通过将左眼的COP设定在(0, 0, −1.5)，将右眼的COP设定在(0, 0, 1.5)实现的。 219

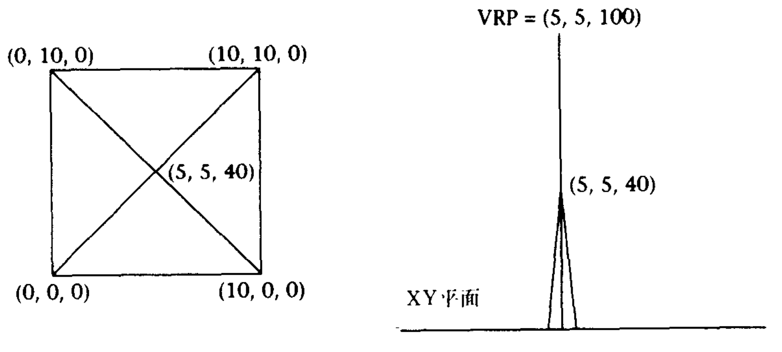


图9-13 对棱锥的观察设置

彩图9-14给出的是视图平面位于棱锥后面的情形。如果两幅图像被融合，那么棱锥应该看起来像是从书页中出来。请看一下书页纹理本身，这将会有助于找到书页相对于虚拟棱锥的位置。注意这对应于负水平视差的情况。

彩图9-15给出的是除了视平面距离变为80之外，其他没有任何变化的图像。所以此时视

平面正好位于棱锥基座与顶点的中间位置。注意现在这个虚拟棱锥是如何开始从书页后面冲出来的。这对应于对象既使用负视差又使用水平视差的情况。

最后，彩图9-16给出了视平面在60处的图像，所以它仅仅触到了棱锥的顶端，视平面上其他点都位于棱锥的前面。注意这时虚拟棱锥看起来多么像在纸的后面。这对应于正视差的情形。

也许这组图最有趣的方面是图9-12中的单一图像与其他几个图像之间的显著差异，以及立体深度线索所传达的大量额外信息。单一视图看起来仅仅像是在一个正方形中摆放的一组二维三角形图案，没有任何线索能表明所描述的是一个3D对象，观察者只能靠想像去把它解释成3D对象。立体视图的深度在另外的几幅图像中是非常突出的。

这个例子所使用的代码如下所示：

```
static Camera_GL *TheCamera;

static Point3D Pyramid[] = {{0.0,0.0,0.0},{10.0,0.0,0.0},
                           {10.0,10.0,0.0},{0.0,10.0,0.0},
                           {5.0,5.0,40.0}};
```

220

```
/*read in command line argument*/
int Eye;            /*- for L, + for R*/
float VPDistance;   /*view plane distance*/

/*preset half inter-pupilary distance*/
#define HIPD  1.5
static void displayPyramid(void)
{
   /*base*/
   glBegin(GL_POLYGON);
      glColor3f(0.0,0.0,0.0);
      glVertex3f(Pyramid[0].x,Pyramid[0].y,Pyramid[0].z);
      glVertex3f(Pyramid[3].x,Pyramid[3].y,Pyramid[3].z);
      glVertex3f(Pyramid[2].x,Pyramid[2].y,Pyramid[2].z);
      glVertex3f(Pyramid[1].x,Pyramid[1].y,Pyramid[1].z);
   glEnd();

   /*front*/
   glBegin(GL_POLYGON);
      glColor3f(1.0,0.0,0.0);
      glVertex3f(Pyramid[0].x,Pyramid[0].y,Pyramid[0].z);
      glVertex3f(Pyramid[1].x,Pyramid[1].y,Pyramid[1].z);
      glVertex3f(Pyramid[4].x,Pyramid[4].y,Pyramid[4].z);
   glEnd();

   /*right*/
   glBegin(GL_POLYGON);
      glColor3f(0.0,1.0,0.0);
      glVertex3f(Pyramid[1].x,Pyramid[1].y,Pyramid[1].z);
      glVertex3f(Pyramid[2].x,Pyramid[2].y,Pyramid[2].z);
      glVertex3f(Pyramid[4].x,Pyramid[4].y,Pyramid[4].z);
   glEnd();

   /*back*/
   glBegin(GL_POLYGON);
      glColor3f(0.0,0.0,1.0);
      glVertex3f(Pyramid[4].x,Pyramid[4].y,Pyramid[4].z);
      glVertex3f(Pyramid[2].x,Pyramid[2].y,Pyramid[2].z);
      glVertex3f(Pyramid[3].x,Pyramid[3].y,Pyramid[3].z);
```

```
    glEnd();

    /*left*/
    glBegin(GL_POLYGON);
      glColor3f(1.0,1.0,0.0);
      glVertex3f(Pyramid[0].x,Pyramid[0].y,Pyramid[0].z);
      glVertex3f(Pyramid[4].x,Pyramid[4].y,Pyramid[4].z);
      glVertex3f(Pyramid[3].x,Pyramid[3].y,Pyramid[3].z);
    glEnd();
}

static void display ()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    displayPyramid();
    glFlush();
}

static void reshape(int width, int height)
{
    setCOP_GL(TheCamera,(float)(HIPD*Eye),0.0,0.0);                       221
    setVPDistance_GL(TheCamera,VPDistance);
    setClipPlanes_GL(TheCamera,0.1,200.0);
    setVPWindow_GL(TheCamera,-10.0,10.0,-10.0,10.0);

    clickProject_GL(TheCamera);
    glViewport (0, 0, width, height); /*define the viewport*/
}

static void initialize(void)
{

    /*GL_FLAT or GL_SMOOTH*/
    glShadeModel(GL_FLAT);

    /*set the background (clear) Color to white*/
    glClearColor(1.0,1.0,1.0,0.0);

    glEnable(GL_DEPTH_TEST);

    /*set the depth buffer for clearing*/
    glClearDepth(1.0);

    /*initialize the camera*/
    TheCamera = newCamera_GL();
    setVRP_GL(TheCamera,5.0,5.0,100.0);
    setVPN_GL(TheCamera,0.0,0.0,-1.0);
    setVUV_GL(TheCamera,0.0,1.0,0.0);

    clickView_GL(TheCamera);
}

int main(int argc, char** argv)
{
    int window;

    glutInit(&argc,argv);

    if(argc != 3){
```

```
    printf("stereo eye(-1 or 1) vpdistance\n");
    exit(0);
  }

222  Eye = atoi(argv[1]);
  VPDistance = (float)atof(argv[2]);

  /*record the window height*/
  Height = 200;

  glutInitWindowSize(Height,Height);

  glutInitDisplayMode(GLUT_RGBA|GLUT_DEPTH);
  if(Eye < 0)   window = glutCreateWindow("Left");
  else
  if(Eye == 0)  window = glutCreateWindow("Mono");
  else          window = glutCreateWindow("Right");

  glutSetWindow(window);

  initialize();

  /*register callbacks*/
  glutDisplayFunc(display); /*display function*/
  glutReshapeFunc(reshape);

  glutMainLoop();
}
```

函数displayPyramid从数组Pyramid中提取坐标并以适当的顺序（反时针方向）构成多边形。函数display清除颜色缓冲区和深度缓冲区，然后渲染棱锥，刷新输出缓冲区。函数reshape设定投影矩阵。注意它设定COP（对于左边或右眼）和视平面距离，以及视平面窗口。COP和视平面距离值是在主函数main（）中通过命令行参数设定的。注意，函数clickProject的调用设定了投影矩阵，如先前小节中所讨论的。函数initialize设定明暗处理类型为单调的（在这个例子中没有光源，所以每个多边形都有一个预先指定的颜色）。它将背景颜色重设为白色（每当颜色缓冲区位被重新设定，如在display中，整个帧缓冲区各处都将被设定为这个颜色）。它激活z深度测试（见第13章），并设置 z 最大值为 1.0（当深度缓冲区位被清除时z缓冲区将设置为此值，如在display中）。该函数之后创建一个新的照相机对象，并设置 VRP、VPN和 VUV值。它还设置先前小节所介绍的矩阵M。

现在我们利用这个机会说明该如何在GLUT系统里嵌入这样一段程序。GLUT是一个实用程序，程序设计者能够使用OpenGL很快地创建交互式程序（OpenGL 只是一个渲染器——它不参与设定显示窗口或交互方面的内容）。关于GLUT将不在本书中介绍，只在例子中使用它。在主函数main中的代码应该是不需要再说明什么了。GLUT使用一个回调系统，所以它在一个主循环中，而且每当某些特殊事件发生时它就调用有关的函数加以响应。在这个例子中，每当需要重新显示帧缓冲区时，它就会调用函数display。每当帧缓冲区渲染的窗口被改变（例如变成可见的了或不再被遮挡了），它将调用函数reshape。

我们鼓励读者去试试这个程序，通过改变一些参数来分析立体视图的效果。

在虚拟现实系统的上下文中，情形将更为复杂。对于头盔显示器，立体系统一定不能只
223 考虑到几何观察系统，而且还要考虑到光学和因透镜所造成的失真问题。 有关这些的详细讨论可以参见Robinett and Rolland（1992），我们在第20章中对此也有一些介绍。对于像 CAVE

这样的系统，那里有多个投影，每面墙壁有一个投影，同样观察系统也一定要考虑到这些，尤其是要考虑在拐角处图像结合在一起的时候会发生什么样的事情。对此的完全讨论可以在Cruz-Neira 等（1993）的著作中找到。

9.10 小结

本章分析了从WC点转换到投影空间的过程，这个过程是为了适应执行透视投影，还介绍了一些深入的计算，如裁剪。由描述视图、视图参照点、视平面法向、视平面上方矢量的参数，我们构造了一个矩阵（M）来执行到观察坐标系的变换。其他一些参数，包括投影中心、视平面距离和视平面窗口，我们用这些参数来构造另外一个矩阵（Q），它负责把VC转变成规范观察坐标系。最后，矩阵P用来转换到投影空间。因此矩阵MQP完成了将WC点转换为 PS点。在下一章中我们将讨论这个过程中更关键的方面：视景体的裁剪过程。

我们使用 OpenGL 研究观察实现的问题。首先看到了基本设施（模型视图矩阵和和投影矩阵堆栈），然后学习了该如何使用GLU实用程序库来定义视图。

最后也讨论了给定观察模型后如何轻松地创建立体视图，并为这个主题的进一步探索提供了一个示例程序。

附录9.1 式(9-8)中矩阵的推导

在式（9-8）中的矩阵将图9-7 的平面变换为比较简单的形式$y=\pm(z+D)$和 $x=\pm(z+D)$。这个变换一定是仿射变换，因为平面性在变换中得到保持。图9-7中最初的平面方程表示为$Dy=y_i(z+D)$和$Dx=x_i(x+D)$（i=1，2）。平面上的每个点$Dy=y_2(z+D)$必须被转换成平面 $y=z+D$上的一个点，对于剩余的三个平面也有相似的结论。而且，在这个变换下距离D没有变，我们知道的一些特殊结果有（_，0，$-D$）→（_，0，$-D$），（_，y_2，0）→（_，D，0）和（_，y_1，0）→（_，$-D$，0），这里“_”表示任意值。对于x同样有相似的结果。很显然，z 一定是不受变换影响的，而且x和y可以被分开单独处理。因此，假如（y，z）被映射为（y'，z'），一定有如下形式：

$$y'=Ay+Bz+C \qquad (9\text{-}27)$$ 224

$$z'=z$$

这里常数 A、B和 C可以通过代入上面三个特殊结果解方程求得。事实上有：

$$A=\frac{2D}{\mathrm{dy}}$$

$$B=-\frac{\mathrm{py}}{\mathrm{dy}} \qquad (9\text{-}28)$$

$$C=-D\left(\frac{\mathrm{py}}{\mathrm{dy}}\right)$$

进而可得到式（9-8）中的矩阵。

附录 9.2 对象层次结构的OpenGL 渲染

本附录给出了使用OpenGL渲染简单对象层次结构的例子。实际上是不能这样使用的——应

该有一个对象层次的数据结构，然后再使用OpenGL进行渲染。这段代码说明了使用OpenGL渲染层次结构的一些简单思想。

```
#include <GL/glut.h>
#include <stdio.h>

/*This next is referred to in several places, and because of the callback
interface, cannot be passed as a parameter, hence, global*/

static void cubebase(void)
/*specifies a side of a cube*/
{
  glBegin(GL_POLYGON);
    glVertex3d(-0.5,-0.5,0.0);
    glVertex3d(-0.5,0.5,0.0);
    glVertex3d(0.5,0.5,0.0);
    glVertex3d(0.5,-0.5,0.0);
  glEnd();
}
```

225

```
static void cube(void)
/*uses cube side to construct a cube, making use of the modelview
matrix*/
{
  /*make sure we're dealing with modelview matrix*/
  glMatrixMode(GL_MODELVIEW);

  /*pushes and duplicates current matrix*/
  glPushMatrix();

  /*construct the base*/
  cubebase();
  glPushMatrix();
  /*construct side on +x axis*/
  glTranslated(0.5,0.0,0.5);
  glRotated(90.0,0.0,1.0,0.0);
  cubebase();

  glPopMatrix();

  /*construct side on -x axis*/
  glPushMatrix();
  glTranslated(-0.5,0.0,0.5);
  glRotated(-90.0,0.0,1.0,0.0);
  cubebase();
  glPopMatrix();

  /*construct side on +y axis*/
  glPushMatrix();
  glTranslated(0.0,0.5,0.5);
  glRotated(-90.0,1.0,0.0,0.0);
  cubebase();
  glPopMatrix();

  /*construct side on -y axis*/
  glPushMatrix();
  glTranslated(0.0,-0.5,0.5);
  glRotated(90.0,1.0,0.0,0.0);
  cubebase();
  glPopMatrix();
```

```
  /*construct top*/
  glPushMatrix();
  glTranslated(0.0,0.0,1.0);
  glRotated(180.0,1.0,0.0,0.0);
  cubebase();
  glPopMatrix();

  glPopMatrix();

}

static void stack(int n)
/*creates a smaller cube on top of larger one*/
{

  cube();
  if(n==0)return;

  glPushMatrix();
  glTranslated(0.0,0.0,1.0);
  glScaled(0.5,0.5,0.5);
  stack(n-1);
  glPopMatrix();
}
static void display()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  stack(6);
  glutSwapBuffers();

}

static void rotate(void)
/*rotates around z-axis*/
{
  static GLdouble a = 0.0;

  /*make sure we're dealing with modelview matrix*/
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glRotated(a,0.0,0.0,1.0);
  cube();
  display();
  glPopMatrix();
  a += 5.0;
}

static void reshape(GLsizei width, GLsizei height)
{
  /*define the viewport - width and height of display window*/
  glViewport (0, 0, width, height);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  /*define view frustrum*/
  gluPerspective(50.0,(GLdouble)width/(GLdouble)height,0.01,10.0);
  /*35deg field of view vertically, with aspect ratio, and
  front and back clipping planes of -1.0 and 10.0*/
}
```

226

```
static void initialize(void)
{
  /*material properties*/
  GLfloat mat_diffuse[] = {1.0,1.0,0.0,0.0};

  /*lighting*/
  GLfloat light_diffuse[] = {1.0,1.0,1.0,1.0};

  /*light position*/
  GLfloat position[] = {1.0,1.0,1.0,1.0};

  /*flat shading*/
  glShadeModel (GL_FLAT);

  /*create normals which are normalized automatically*/
  glEnable(GL_NORMALIZE);
  glEnable(GL_AUTO_NORMAL);
  /*set the background (clear) Color to white*/
  glClearColor(1.0,1.0,1.0,0.0);

  /*enable lighting*/
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  /*for 2D the modelview matrix is the identity*/
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(3.0,3.0,4.0,/*eye*/
           0.0,0.0,0.0,/*looking here*/
           0.0, 0.0, 1.0);/*up vector*/

  /*set the light position in eye (viewing) coordinates*/
  glLightfv(GL_LIGHT0,GL_POSITION,position);
  /*actually this is direction, since by default an infinite
  light source is assumed*/

  /*set the material*/
  glMaterialfv(GL_FRONT,GL_DIFFUSE,mat_diffuse);

  glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);

  /*enable the depth buffer*/
  glEnable(GL_DEPTH_TEST);

  /*set the depth buffer for clearing*/
  glClearDepth(1.0);

}

int main(int argc, char** argv)
{
  int window;

  glutInit(&argc,argv);

  glutInitWindowSize(500,500);
  glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_DEPTH);

  window = glutCreateWindow("House");
```

227

```
  glutSetWindow(window);

  initialize();

  /*register callbacks*/
  glutDisplayFunc(display); /*display function*/
  glutReshapeFunc(reshape);
  glutIdleFunc(rotate);

  glutMainLoop();
}
```

228

第10章　裁剪多边形

10.1 引言

在前一章中我们考虑了变换序列，通过变换序列将对象局部坐标系中的一个点转换成在显示器上的投影。这样的一个投影是经过虚拟照相机获得的。这个观察管道的一个非常重要的方面先前并没有考虑到，那就是**裁剪**。在光线跟踪中这不是什么问题，因为没有主光线会跑到视景体的外面，这个视景体区域由四个分别通过视平面一条边的平面和前后裁剪平面所包围。然而，在新的方法中，也就是当我们将多边形投影到视平面上时，必须明确地执行对多边形的裁剪，以便只考虑位于视景体里面的对象或对象的一部分。在这一章中我们将着重研究多边形的裁剪问题。

我们知道多边形是一个点序列 $[p_0, p_1, \cdots, p_{n-1}]$，$p_n \equiv p_0$，相邻点用直线段连接，$p_{n-1}$与$p_0$首尾相连。$p_i$称为多边形的**顶点**，连接顶点间的直线段称为**边**。我们假设所有多边形都是在平面上的（所有的顶点在同一个平面上）。如果一个多边形的边除了在公共顶点处外彼此不相交，那么它称为**简单多边形**。如果一个多边形的每一个内角都小于180度，也就是说两条边在多边形里侧所夹的角度小于180度，那么它称为**凸多边形**。在这一章中我们所考虑的都是简单多边形。

为了维护基本类型的一致性，多边形的裁剪结果也应该是一个多边形（或者什么也没有，如果多边形完全在裁剪区域的外面）。如果在裁剪之后最初的多边形变成了支离破碎的一些线段，那么所设计的在显示器上渲染多边形的算法就不能继续使用了。因此经过裁剪，原始多
229 边形的边将被裁剪区域的边界所代替。

我们首先考虑二维中多边形的裁剪，然后在先前一章所描述的空间上下文中将它扩展到三维。

10.2 Sutherland-Hodgman 算法（二维）

这一小节我们讨论在二维中将一个多边形裁剪到一个矩形窗口。在二维中的一般问题是将一个多边形（被裁剪者）裁剪到另一个多边形（裁剪区域）中。这两个多边形当然是在相同的平面上。裁剪区域的边我们称之为**边界**。

最后结果是一个新的多边形（可能为空），该多边形包含了被裁剪者和裁剪区域的公共部分。这个新多边形的每条边可能是原始多边形的一条边，或者是一条边的一部分，也可能是裁剪区域的一条边或其一部分。在这里我们只考虑裁剪区域为其边与坐标轴平行的矩形，但是所讨论的算法是适合于更复杂的区域。

由Sutherland和 Hodgman（1974）发明的多边形裁剪过程是这样的，先用裁剪区域的一个边界，比如说上边界来裁剪多边形。所产生的多边形再用裁剪区域的第二个边界裁剪，比如说右边界来裁剪。再将结果分别用下边界和左边界进行裁剪。该方法如图10-1所示。

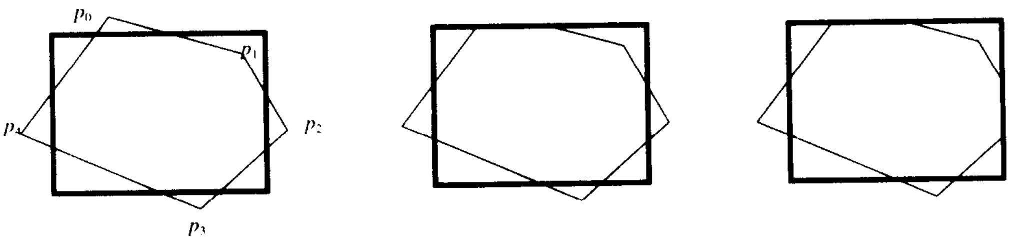
用边界裁剪一个多边形是简单的。过程是对多边形边进行迭代，构造多边形一个新的顶点序列，用它来代表经过裁剪之后的多边形。设这个新的顶点序列（P）被初始化为空。那么，对于每条边（在这里表示为p_0到p_1），有四种可能情形需要考虑:

（1）由p_0到p_1的线段进入了裁剪区域（也就是说，p_0在相应的裁剪边界的外面，而p_1已经在裁剪区域的里面）。在这种情形，设p是线段p_0到p_1与裁剪边界的交点，将p点和p_1点依次送入P的尾部。

（2）线段在离开裁剪区域（也就是说，p_0在裁剪区域的里面，而p_1在外面）。同样，我们设p是线段和裁剪边界的交点，将p点送入P的尾部。

（3）线段完全落在裁剪区域的外面——此时什么也不做。

（4）线段完全落在裁剪边界的可见一侧——在这种情况下，将p_1点送入P的尾部。 230



a) 裁剪一个多边形　　b) 裁剪上侧　　c) 裁剪右侧

图 10-1

(1)~(4)步在图10-2中说明。对多边形每条边依次执行完这些步骤之后，序列P将包含由相关边界裁剪过后的多边形顶点。(1)~(4)中的测试可以很容易完成。假设所考虑的边界是那条“左边”，其方程为 $x=xmin$。那么对于多边形边$p_0(x_0, y_0)$到 $p_1(x_1, y_1)$有：

$$
\begin{aligned}
&x_0<xmin,\ x_1>xmin \rightarrow (1)\\
&x_0>xmin,\ x_1<xmin \rightarrow (2)\\
&x_0<xmin,\ x_1<xmin \rightarrow (3)\\
&x_0>xmin,\ x_1>xmin \rightarrow (4)
\end{aligned}
\qquad (10\text{-}1)
$$
231

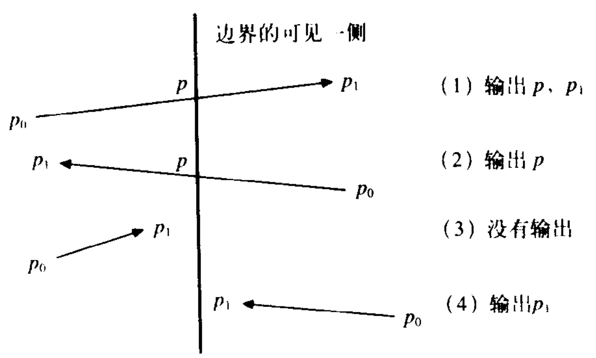


图10-2 用边界裁剪一多边形

10.3 裁剪多边形——Weiler-Atherton 算法

考虑图10-3中的多边形B。SH算法的应用所产生的裁剪多边形的顶点为i，b，l，4，5，k，b，j，9，0——这个输出会成为一个带有“退化”边的多边形，也就是说输出多边形的一些边会发生重叠现象。注意，依照我们的最初定义这仍然是一个有效的多边形——它是顶点序

列且是封闭的。同时，这样一个多边形可以被多边形渲染程序正确地处理。然而，有例子表明退化边会成为一个问题。举例来说，在一些隐藏面和阴影检测算法中需要裁剪，在那里多边形 B 的输出最好是两个不同的多边形。

Weiler 和 Atherton（1977）构造了一个算法，允许对任何简单多边形在任何其他多边形所构成的裁剪区域裁剪。我们还是在矩形裁剪区域的上下文中来讨论这个算法。它基于这样的思想：在输出多边形中的边片段要么是已经在输入多边形中的，要么是裁剪区域边界中的。而且，裁剪区域边界的任何落在多边形内部的部分都将多边形分割成里和外两个区域。裁剪区域边界的这些部分总是开始和结束于多边形和裁剪区域边界的交点。

232

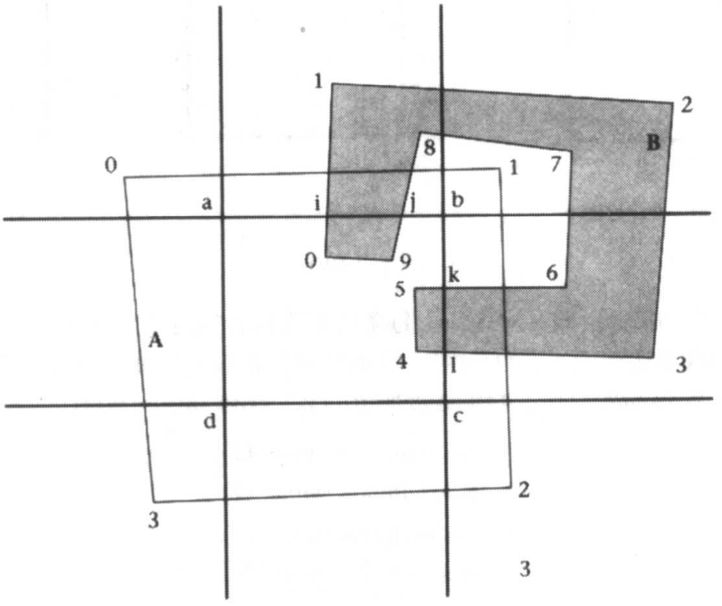


图10-3 用多边形A 裁剪多边形B

我们假设多边形和裁剪区域的遍历顺序是一致的——比如是顺时针。第一步是计算多边形的边和裁剪区域边界的所有交点。在多边形B中，这个结果是集合 {i，j，k，l}。每一个这样的相交顶点被分为两类，一类是多边形进入裁剪区域（E）的点类型，一类是离开裁剪区域（L）的点类型。在这个例子中所分成的两个集合是 L: {i，k} 和 E: {j，l}。

我们需要一个用来表示原始多边形顶点和裁剪区域顶点的数据结构。这个数据结构应该由两个序列组成，但是在序列中最后一个点应该有指向第一个点的指针——表示两集合中的顶点是顺时针方向排序的。最后相交顶点必须在两个序列中建立索引，以便在一个相交顶点处可以从多边形顶点列表遍历到裁剪顶点列表，反之亦然。因此我们可以沿着多边形顶点和裁剪区域顶点跟踪两个完整的边线。一个例子如图10-4 中所示。

现在我们从集合E开始，选择其中的一个点，比如j点。因为在j点多边形的路径是进入裁剪区域，我们沿着多边形路径向前直至遇到下个相交点。这样得到了一个序列9，0，i。下个顶点i是集合L的，因此沿着裁剪路径前进，它会把我们带回到j点。这样就完成了这个多边形的输出。现在让我们回到集合E并选择另外的一个顶点。这时只有一个顶点存在，即顶点l。这样得到序列 4，5，k。最后是一个离开顶点，我们沿着裁剪路径前进，它把我们带回到顶点k。这样就完成了第二个多边形。现在集合E 变成空的了，我们完成了全部任务。

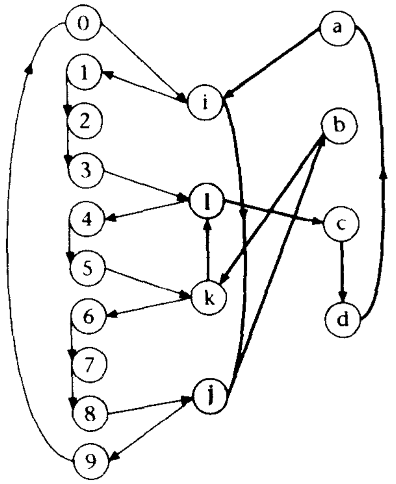


图10-4 相交顶点插入的数据结构：粗线路径表示沿着裁剪区域顶点的序列。其他路径是沿着多边形顶点的序列。E的相交顶点都位于阴影圆中，L的相交顶点都位于无阴影的圆中 233

总结：

- 对多边形和裁剪区域顶点建立环形链表。
- 求出边和边界的所有相交，将相交顶点放入列表中，这样就可以在相交顶点处从一个链表遍历到另一个链表。
- 分割相交顶点集合为集合E和集合L。集合E包含的顶点是多边形从外部进入裁剪区域的那些顶点，集合L所包含的顶点是当多边形离开裁剪区域时的那些顶点。

```
while (E is not empty){
   select and remove v from E;
   start a new empty polygon P;
   initialize w = v;
   do{
      w = nextPolygonVertex(w); append(P,w);
      while(w ≠ intersection vertex){
         w = nextPolygonVertex(w);
         append(P,w);
      };

      w = nextClipVertex(w); append(P,w);
      while(w ≠ intersection vertex){
         w = nextClipVertex(w);
         append(P,w);
      };
   } while(w ≠ v);
}
```

注意这个算法依赖函数nextPolygonVertex和函数nextClipvertex，它们分别返回多边形顶点列表中的下一个顶点和裁剪区域顶点列表中下一个顶点。算法输出一个多边形，它是集合E中每个顶点所构成的。

Weiler(1980)给出了这个算法的一个改进算法，它采用一个比上面数据结构更复杂的数据结构。我们注意到算法中的一个主要困难是计算相交值并且在顶点链表中准确地存储它们。可以证明使用平面扫描算法将更为高效（Preparata and Shamos，1985）。

10.4 在三维中裁剪多边形

Sutherland-Hodgman 算法推广到三维是很容易的。然而，在做之前，我们必须首先准确识别出三维裁剪边界。在规范观察空间中，视景体是一个规则金字塔由前裁剪平面和后裁剪平面切割所成的形状。在投影空间中视景体是一个立方体。首先我们考虑在投影空间中裁剪到视景体的情况。

234 接着，我们将重点考虑在规范观察空间中的裁剪。最后还要考虑在齐次坐标空间中裁剪的优势，所谓的齐次坐标空间是应用式（9-16）中的矩阵和式（9-14）中的P而定义的。

通常三维的裁剪算法与二维的算法有相同的结构。细微变化是：

- 裁剪边界是平面，共有六个这样的平面。
- 线段或边与这些平面之间的相交计算显然是不同的。

投影空间中的裁剪

投影空间中的视景体是由下式定义的：

$$\begin{aligned} -1 \leqslant x \leqslant 1 \\ -1 \leqslant y \leqslant 1 \\ 0 \leqslant z \leqslant 1 \end{aligned} \tag{10-2}$$

六个裁剪平面和参数化定义的直线的相交t值在表10-1中给出。

表10-1　投影空间的裁剪平面和交。考虑直线段：（x_0，y_0，z_0）到（x_1，y_1，z_1）

平面名称	投影空间中的平面方程	相交所对应的t值
235 左	$x=-1$	$-\left(\frac{1+x_0}{dx}\right)$
右	$x=1$	$\frac{1-x_0}{dx}$
底	$y=-1$	$-\left(\frac{1+y_0}{dy}\right)$
顶	$y=1$	$\frac{1-y_0}{dy}$
前	$z=0$	$-\frac{z_0}{dz}$
后	$z=1$	$\frac{1-z_0}{dz}$

Sutherland-Hodgman 多边形裁剪算法是针对每个边界去裁剪整个多边形，然后将所得到的多边形（如果存在的话）传递给下一个边界。这需要两个操作：

（1）确定多边形边和边界之间的关系（图10-2）：

- p_0 在内部且p_1在内部
- p_0在内部而p_1在外部
- p_0在外部而p_1在内部
- p_0在外部且p_1在外部

(2) 求出多边形边和边界之间的交。

考虑$x=1$（右）裁剪平面，(1) 中的四个条件将是：

- $x_0<1$和$x_1<1$
- $x_0<1$和$x_1>1$
- $x_0>1$和$x_1<1$
- $x_0>1$和$x_1>1$

对于其他几个平面的情况是相似的。现在我们能根据在二维中的算法内容和上述的一些具体变化构造SH算法。

规范观察空间中的裁剪

有些时候在PS中裁剪会产生不正确的结果。图10-5给出了一种情况，这里有一条p_1到p_2的线段，p_1位于COP的后面，p_2位于COP的前面。在视平面上该直线的投影实际上产生了两条无限直线，如图所示。看一下p_1的投影——从通过COP点投影在视平面上的q_1点。由于它在COP的后面，对投影没有产生影响（根据定义，一个点在视平面上的投影是从该点出发经过COP点的光线与视平面的相交点）。 236

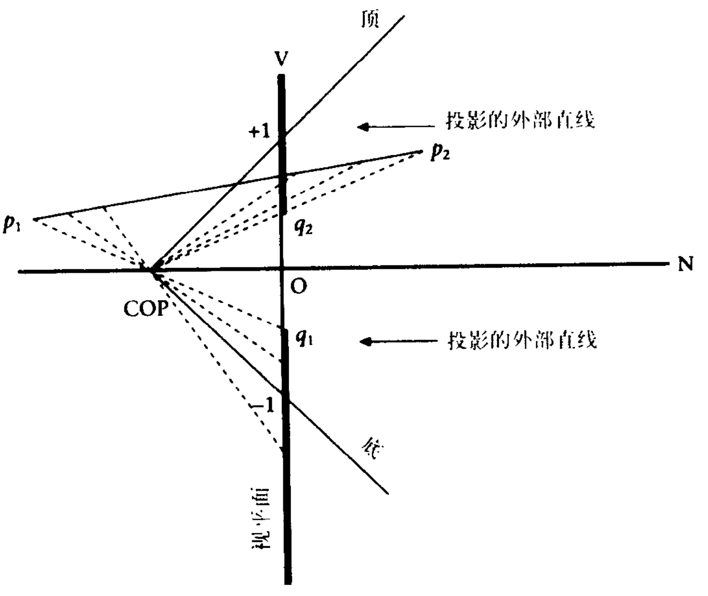


图10-5 在规范观察空间中的直线p_1到p_2的裁剪

假若点不在平行于视平面且过COP点的平面上，则投影是存在的。现在如果选择靠近p_1但仍在线段上的另外一个点，这将投影到视平面的另外一个点上。如果以这种方式继续下去，将生成一条无限长的直线（在视平面上较低的那个）。同样地，如果从p_2点开始，经过COP的光线与视平面相交于q_2点，然后选择靠近p_2点的其他点由此生成了第二条无限直线（图中视平面上靠上的那个）。因此对单条线段的正确投影被称为“外部直线”。

这种情形将导致一个严重的错误，因为一般来讲线段的两个终点p_1和p_2投影在视平面上后，会在视平面上形成从q_1到q_2的线段。然而连接q_1和q_2的线段是真正直线的补，在例子中，正确的结果应该是从“负无穷”到q_1以及从q_2到“正无穷”。这好像很奇怪，但在数学上是正确的。

在现实视觉中它并不会发生，这是因为在我们眼睛后面的部分场景已经被我们的头部“裁剪”掉了。同样地，如果我们在COP前面加上一个前裁剪平面，所有在 COP 之后的场景部分将会被删除，这种情形不会出现。

我们知道转换到投影空间等价于执行投影（举例来说，曾经在 PS 中我们所处的观察情形等同于正交平行投影——所以透视投影一定已经执行过了）。因此，每当有部分场景在COP 后面的时候，转换到PS然后裁剪将会导致不正确的结果。换句话说，每当 COP 位于所观察的场景之中，先投影后裁剪是不正确的。交互式漫游应用中 COP 会在场景中处处移动，所以对于这种类型的应用，在PS中裁剪是错误的。

然而，这不是一个问题。我们可以很容易地在规范观察空间中执行裁剪，用不等式定义如下：

$$\begin{gathered}-(z+1) \leqslant x \leqslant z+1 \\ -(z+1) \leqslant y \leqslant z+1 \\ Dmin \leqslant z \leqslant Dmax\end{gathered} \tag{10-3}$$

同样裁剪到这个空间是很容易的，几乎不需要对 SH 算法做什么变化。具体的变化是点相对于边界的可见性测试和相交测试。各种平面方程和相交的t值在表10-2中给出。

举例来说，考虑其中一个裁剪平面：$y=z+1$。代入直线的参数化方程，求出在与平面相交点处的参数 t：

237

$$\begin{gathered}y(t)=z(t)+1 \\ y_0+t\mathrm{d}y=z_0+t\mathrm{d}z+1\end{gathered} \tag{10-4}$$

表10-2　规范观察空间的裁剪平面和交。考虑直线段：(x_0, y_0, z_0) 到 (x_1, y_1, z_1)

平面名称	投影空间中的平面方程	相交所对应的t值
左	$x=-(z+1)$	$-\left(\frac{z_0+x_0+1}{\mathrm{d}x+\mathrm{d}z}\right)$
右	$x=z+1$	$\left(\frac{z_0-x_0+1}{\mathrm{d}x-\mathrm{d}z}\right)$
底	$y=-(z+1)$	$-\left(\frac{z_0+y_0+1}{\mathrm{d}y+\mathrm{d}z}\right)$
顶	$y=z+1$	$\left(\frac{z_0-y_0+1}{\mathrm{d}y-\mathrm{d}z}\right)$
前	$z=Dmin$	$\frac{Dmin-z_0}{\mathrm{d}z}$
后	$z=Dmax$	$\frac{Dmax-z_0}{\mathrm{d}z}$

整理得：

$$\begin{gathered}t(\mathrm{d}y-\mathrm{d}z)=z_0-y_0+1 \\ \therefore t=\frac{z_0-y_0+1}{\mathrm{d}y-\mathrm{d}z}\end{gathered} \tag{10-5}$$

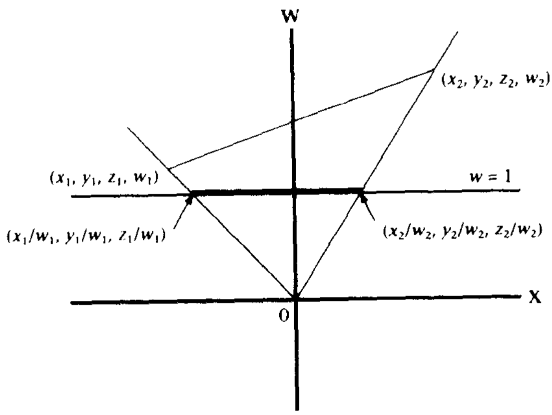
将它代入参数化直线方程，求出相交点。

齐次空间中的裁剪

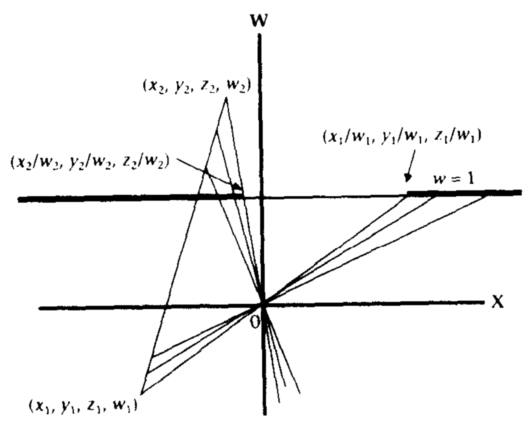
WC点表示为一个4D齐次点，然后乘以矩阵T（参见式（9-16）），将它转换到规范观察空间。现在假设我们不是在那里裁剪，而是继续使用投影矩阵（参见式（9-14））。它将点转变成一般的齐次形式(x,y,z,w)，这里在PS中等价的3D点是（X，X，Z），且有：

$$X=\frac{x}{w},Y=\frac{y}{w},Z=\frac{z}{w},w\neq 0 \qquad (10\text{-}6)$$ 238

除以w事实上是将4D的点投影到超平面w=1上，如图10-6所示，一条直线被投影到超平面w=1上。图10-7说明了如果一条直线其中一个端点具有负的w值，那么直线投影到外部直线段上——也就是说，它"交汇"在无限远处。我们注意到在w=0平面上的点将投影到无限远（它们没有在w=1上的投影）。



映射直线（x_1、y_1、z_1、w_1）到（x_2、y_2、z_2、w_2）



映射直线（x_1、y_1、z_1、w_1）到（x_2、y_2、z_2、w_2）

图10-6 从齐次坐标到3D坐标的变换：一条内部直线段 图10-7 从齐次坐标到3D坐标的变换：一条外部直线段

当投影矩阵P以如下所示的值应用于规范观察空间时，一件有趣的事情发生了：

$$P=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1\\0&0&0&1\end{bmatrix} \qquad (10\text{-}7)$$ 239

如图10-8所示。位于（0，0，－1，1）的COP点将被转换到w=0平面。因此在观察空间中的直线如果有一个点位于COP的后面，它将会产生外部直线段投影。Blinn和Newell（1978）说明这个问题可以通过裁剪过程克服，但是很关键的一点是裁剪要在从齐次空间到三维PS空间的变换进行之前完成（也就是说，在用w除之前）。因此，不要先转换到PS空间然后再裁剪，而是先裁剪然后除以w。这与上面所处理的问题是一样的，只不过是在不同的空间中考虑的。因而裁剪可以在四维空间中进行，最好不要在规范观察空间中进行。

在四维空间中的裁剪与在规范三维观察空间中的裁剪其执行是相似的。然而，裁剪边界的表示不一样。举例来说，三维PS的边界有如下形式：

$$-1<\frac{x}{w}<1 \qquad (10\text{-}8)$$

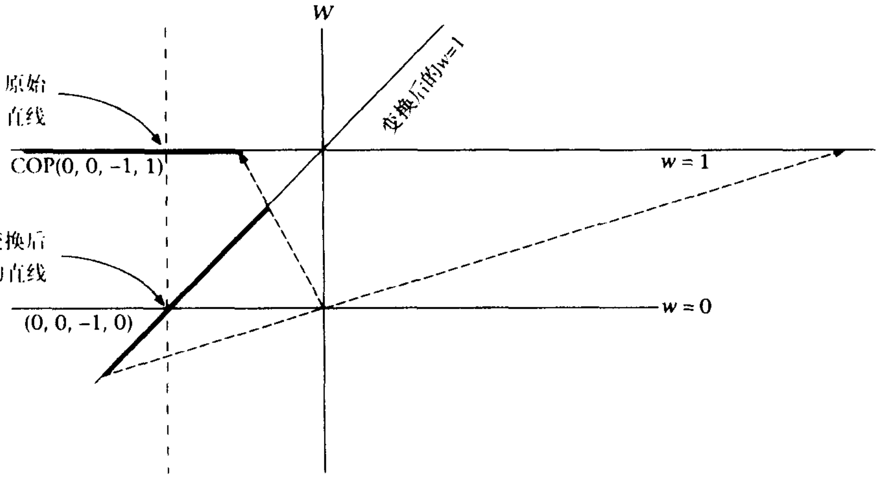


图10-8　COP的变换

等同于

$$-w \leqslant x \leqslant w(w>0)\text{或}$$

240　$$w \leqslant x \leqslant -w(w<0) \qquad (10\text{-}9)$$

这可以用图10-9说明。用这些边界的裁剪与在所有其他空间中的裁剪是同样容易的。然而，在齐次坐标中考虑问题显然会出现外部直线段，这个问题很容易产生（如果应用程序需要的话）。因此，在这个层次上裁剪会有前裁剪平面位于COP后面的可能性（虽然对此的物理解释还不太清楚）。

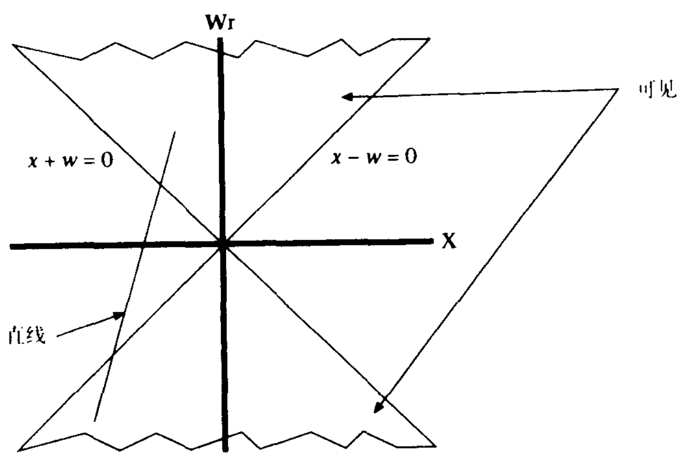


图10-9　四维空间中的裁剪区域

因此裁剪区域是通过下列不等式定义的：

$$-w \leqslant x \leqslant w(w>0), w \leqslant x \leqslant -w \ (w<0)$$
$$-w \leqslant y \leqslant w(w>0), w \leqslant y \leqslant -w \ (w<0) \qquad (10\text{-}10)$$
$$0 \leqslant z \leqslant w(w>0), w \leqslant z \leqslant 0, \ (w<0)$$

求交很容易：考虑平面$x=w$（即$w-x=0$）和下列齐次坐标中的直线

$$p(t)=(x_1,y_1,z_1,w_1)+t(\mathrm{d}x,\mathrm{d}y,\mathrm{d}z,\mathrm{d}w) \tag{10-11}$$

那么相交发生的时候有:

$$(w_1+t\mathrm{d}w)-(x_1+t\mathrm{d}x)=0$$
$$\therefore t=\frac{x_1-w_1}{\mathrm{d}w-\mathrm{d}x} \tag{10-12}$$ 241

相交点可以通过将上述t值代入直线方程求得。其他相交计算类似。表10-1和表10-2的等价构造留给读者作为练习。

在图10-7中有一条直线段穿过$w=0$平面。对这样一个线段的裁剪应该产生两个输出线段。这可以通过下面分析得到:

- 如果直线的两个端点都有$w>0$，那么如通常一样裁剪直线。
- 如果直线的两个端点都有$w<0$，那么关于$w=0$平面镜像该直线，并如通常一样裁剪。
- 如果直线的一个端点$w<0$，而另一个端点$w>0$，那么首先裁剪直线到裁剪区域的正的部分，然后关于$w=0$平面镜像直线并重复。

如果按照这一步骤，两个输出线段将会在所需的情况下正确地产生。进一步的细节可以从Blinn.op.cit.中看到。在那我们可以看到当所描述的情形自然发生的时候，对一个参数化曲线裁剪的例子。

当裁剪区域包括前裁剪平面和后裁剪平面时，且所裁剪的对象为直线和多边形，就可以避免两个裁剪（$w>0$和$w<0$）。这里，前裁剪平面位于COP的前面。用参数z表示的裁剪区域是$0<z<w$。因此，如果先用$z>0$和$z<w$进行裁剪，那么经过裁剪的点一定有$w>0$。因此，举例来说，当使用 Sutherland-Hodgman 算法的时候有:

- 裁剪到$z=0$。
- 裁剪到$z=w$。
- 以任何顺序裁剪到剩余的平面。

10.5 小结

本章完成了对观察管道的介绍，说明了裁剪过程是如何在任意的照相机位置和方向设置下与变换结合的。我们只是在一个很有限的范围内讨论了裁剪问题：在二维中只讨论矩形窗口的裁剪，在三维中只讨论由六个裁剪平面所围成的金字塔形视景体的裁剪。而且，我们只处理多边形的裁剪。在第17章中我们将会继续讨论直线的裁剪问题。

回到总体渲染管道，我们现在有能力渲染由多边形所构造的层次场景，允许对于任何照相机参数设置。这个过程比光线跟踪中的最初起点要快得多，因为我们是将多边形投影到视平面上而不是执行数以百万计的光线-多边形相交测试。当然，我们也失去了光线跟踪容易处理的对象间交叉反射。我们可以为了速度牺牲真实感，然而不能够牺牲掉光线跟踪能轻易处理的其他东西——那就是可见性。如果在处理整个场景图时只是简单地对所遇到的每个多边形执行渲染的话，结果将会是非常错误的。必须考虑到可见性关系，就像通过当前照相机装置去“看”一样。找到一个方式排序多边形以便能根据可见性对它们进行正确的渲染——那些“远处”的多边形先于近处的多边形被渲染，这样比较近的多边形就会覆盖远处的那些多 242
边形。可以使用一个特别的数据结构“二叉空间分割树”完美实现——这是下一章的主题。这个数据结构允许我们再导入阴影——这也是不采用光线跟踪所失去的东西。

第11章 可见性确定

11.1 引言

在前一章中，我们学习了如何创建一个场景并通过虚拟照相机去观察那个场景。这是通过将场景的每个多边形投影到二维图像平面上完成的。在这一章中我们要介绍可见性计算问题。这可以分为两个部分，其中一个部分是我们已经讨论过了的。只有落在视景体中的对象才是看得见的——在先前的一章中我们知道了该如何使用裁剪来达到这一点。然而，一个更有意义也是很困难的问题是对象间的可见性问题。从任何视点看，都会存在一些对象遮挡了其他一些对象，而且当你移动头部的时候，对象被遮挡的部分就会发生变化，同时会有新的部分进入视野。因此如果我们只是简单地将每个多边形按照场景数据库中所给定的顺序来投影，其结果通常是不正确的。这没有考虑可见性，将会导致场景渲染的严重错误。确定图像每个区域中每个对象的可见部分问题称为*可见表面确定*问题。这方面的经典参考书当推Sutherland et al.（1974）。

处理对象间可见性有许多方法，它们大致可以被分为三类。*对象精确方法*在对象彼此之
243 间进行比较，从而确定每个对象在图像中可见的部分。这一类中的第一个例子是由Weiler和Atherton给出的（1977）。他们使用了一般的裁剪方法，用离视点较近的多边形边界去分割那些远离视点的多边形，丢弃它们被覆盖的区域。对象精确算法可以被认为是一种连续求解方法（直到超出机器的精度允许范围），但是当环境尺寸变大时通常存在可量测性方面的问题，而且在实现方面很难达到鲁棒性。*图像精确方法*不同，它处理图像的离散表示。总的思想是通过确定图像每个像素上的可见对象给出在图像分辨率上的一个解。光线投射是这种类型的一个例子。另外的一个例子是我们将会在第13章中看见的z缓冲区方法。这些方法看起来容易实现得多，而且更鲁棒，因而非常流行。最后是第三种类型，也是我们在这一章中将要讨论的，它是一种结合对象精确和图像精确运算的混合方法。

特别地，我们将要介绍一种所谓的*列表优先权算法*。其背后的思想如下：我们设法确定所有的多边形顺序，对任何多边形P和Q，如果Q遮挡P的某一部分，则P将会排在Q的前面。然后我们按照这个顺序渲染多边形，那些遮挡别的多边形的多边形将在后面渲染，这样它们的渲染结果自然覆盖了那些被遮挡的部分。这方面一个不十分精确但是很有用的方法是根据离视点远近来确定渲染顺序，"比较远的"多边形先于"比较近的"多边形渲染。我们称这种覆盖方法为*画家算法*，因为它与画家绘画很相似，画家总是首先画背景，然后再画前景中的对象。算法的最困难部分是正确地对多边形排序。

在这一章中我们将首先考虑封闭多面体的可见性问题，并介绍背面删除的概念。接下来我们将考虑一些列表优先权算法。深度排序算法首先将所有多边形投影到投影空间中，然后得到在这个空间中的排序。注意，这是一种视图依赖方法，因为排序是在投影之后执行的，因此当视图一改变，它就需要再一次从头开始执行。最后我们考虑一些视图独立的解法，即基于对象的分割树和"二叉空间分割"（BSP）树。它们构造在世界空间中的数据结构，由此

可以非常容易地针对给定视点构造一个序列。我们还要说明当维持一个正确的 BSP 树时对象是如何动态地改变的。

11.2 背面删除

背面删除可以使用在场景由一组平面多边形表示，而所有对象都是封闭多面体的情况下。多面体的每个面要么是朝向照相机的，要么是背向照相机的。那些背向照相机的面可以在进一步的计算中不再考虑。很容易计算一个面是否是背向照相机的，这可以从多边形的平面方程得到确定。 244

假定我们使用右手WC坐标系，选择多边形的三个连续顶点，使得它们按反时针方向排序，即当我们从多边形的外部（也就是说从它的前面）观察时，它们是反时针方向的。那么，平面方程将是$l(x, y, z)=ax+by+cz-d=0$，如第8章中所讨论的。

这样的多边形的前向法向量是(a, b, c)，它指向多边形所希望的“外侧”（前向）。如果这个向量背离照相机，那么这个多边形将是背向的，将被删除掉。

假如 $COP=(c_x, c_y, c_z)$是COP点在WC中的表示，那么如果(c_x, c_y, c_z)在多边形平面的前方，则$l(c_x, c_y, c_z)>0$；反之，若是在多边形平面的后面，则$l(c_x, c_y, c_z)<0$。

另外一种做法是假设我们已经将多边形转换到了投影空间。那么如果法向（在PS中计算的）的z单元是正值，该法向一定背离观察者（请记住COP位于这个空间的$(0, 0, -\infty)$处）。然而，这种方法需要在 PS 中计算平面方程。这可以使用经转换的多边形三个连续点直接进行，或从最初的WC平面方程计算得到。平面方程如下：

WC 平面方程可以被表示成：

$$p \cdot l=0 \tag{11-1}$$

这里 $p=(x, y, z, 1)$是平面上的任意点，$l=\begin{bmatrix} a \\ b \\ c \\ -d \end{bmatrix}$。

假设S是一个转换矩阵，对平面上的每个点进行变换。因此对任何点 p，设

$$q=pS \tag{11-2}$$

是转换后的点。

用S^{-1}右乘得到：

$$qS^{-1}=p \tag{11-3}$$

用l右乘并使用式（11-1），有：

$$qS^{-1} \cdot l=p \cdot l=0 \tag{11-4}$$

最后，令 $m=S^{-1} \cdot l$，有：

$$q \cdot m=0 \tag{11-5}$$ 245

这个推导的意义在于如果l表示在某个空间中的平面方程，那么在那个空间中的对象通过矩阵S转换，m代表转换后的平面方程。

很有趣的一点是，对于单个的凸对象，背面删除足以解决可见性问题。根据定义，对于

一个凸的对象，当任意给定光线穿过它时，顶多会产生两个交点（当光线与对象的边或面相切时只有一个交点）。

很容易证明对于两点的情形，最靠近的一个总是前向的，而另一个则是背向的，由此可以测试多边形是否是背向的，若不是则可以以任何顺序渲染它（参见图11-1）。

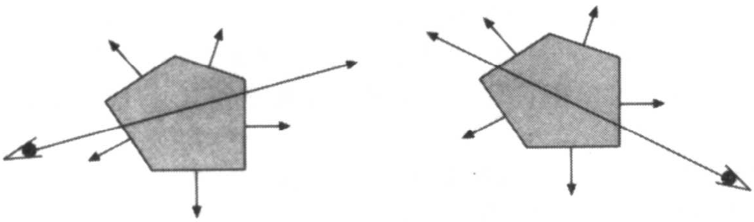


图11-1 对于封闭的凸对象，只要使用背面删除算法，任意多边形的顺序都是有效的

11.3 列表优先权算法

如我们已经提到的，列表优先权算法总的思想是相对于COP点“由远及近”地对多边形排序，并使用画家算法进行渲染。

让我们具体地看一下所说的这个排序的真实含义。假设环境中有*n*个多边形，我们想要一个序列｛$P_1 \cdots P_n$｝，保证任何多边形P_i不会遮挡｛$P_{i+1} \cdots P_n$｝中的任意一个多边形。直观上我们可以认为多边形P_i比｛$P_{i+1} \cdots P_n$｝离COP点更远一点，因此要先于那些多边形渲染。然而我们将会很快发现距离准则是不充足的。

在投影空间中排序

这个思想的最简单实现是所谓的z排序。多边形一经转换到投影空间，我们就求出每个多边形的中心点到COP的距离（这里只是使用中心点的z 坐标），然后在一维中根据这个距离进行排序。在低维空间中排序是高效的，但是
246 并不总是正确的。类似的思想在z缓冲区方法广泛使用之前经常使用在游戏中，用于当视点作小步移动时对不可预知的对象的出现和消失做判定。为了加深对这一点的理解，让我们看一下图11-2的例子。当COP从COP1移动到COP2 时，我们从$d_Q<d_P$变化到 $d_Q>d_P$，所以在一些点处，Q将会在P之前渲染，而Q将从视图中消失。

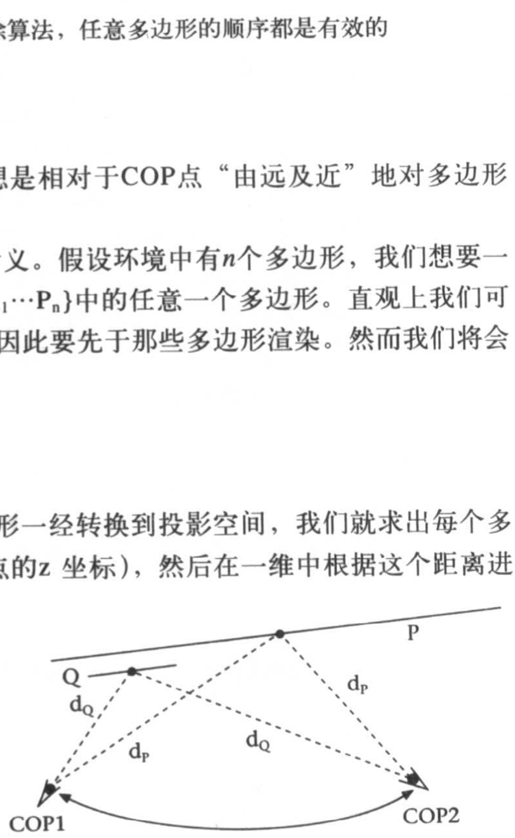


图11-2 当我们从COP1移动到COP2时，多边形顺序在改变

我们可以使标准更严格一些来避免这样的错误发生。对于一对多边形 P、Q，我们不去比较它们中点的各自距离，而是检查在Z方向上它们是否重叠。如果$Z_{maxQ}<Z_{minP}$或者$Z_{maxP}<Z_{minQ}$，那么显然它们中的一个定位于另一个的前面。但是如果不是这样，可能是两个多边形没有投影在图像的相同区域上，在这种情况下它们之间的排序可以是任意的。测试它们是否投影在相同区域的一个简便方法是沿着X轴和Y轴的方向比较多边形的范围（如图11-3）。

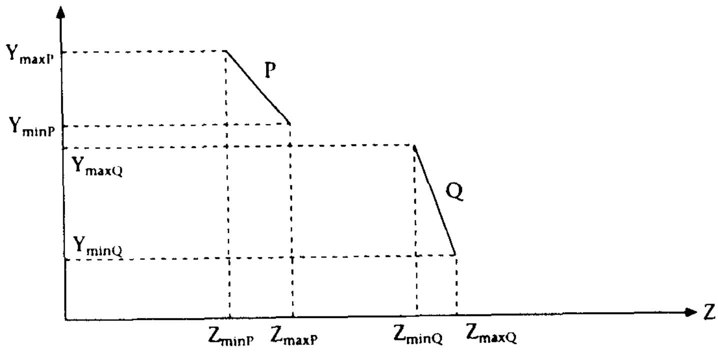


图11-3 P的范围和Q的范围没有Z或Y的重叠

深度排序

深度排序算法（Newell、Newell and Sancha，1972）使用上述思想对多边形排序。这个方法由于其历史原因而变得十分有趣，而且思想是很重要的，尽管今天该算法已经不再实际使用了。假设有两个多边形P和Q，如果下列测试中任何一项成功的话，P可以先于Q渲染。测试顺序如下所示：

（1）Q的Z范围完全位于P的Z范围之前。

（2）Q的Y范围与P的Y范围不重叠。 247

（3）Q的X范围与P的X范围不重叠。

（4）P上的所有点相对于平面Q来说与视点位于相反的两侧。

（5）Q上的所有点相对于平面P来说与视点位于相同的一侧。

（6）P和Q在XY平面上的投影不重叠。

算法如下：

- 根据多边形的最大Z值对它们进行完全排序，设P为这个列表中的最后一个。
- 设Qset是这样的一个多边形集合，其中多边形的Z范围与P的Z范围发生重叠。如果Qset为空，那么渲染多边形P。
- 对在Qset中的每个Q，应用上述的测试（2）～（6）直到有一个成功或者全部失败。如果有一个成功，则渲染P。
- 对任何所有测试都失败的Q，令其与P交换位置（也就是说让它担任先前由P所承担的角色），标记Q为已经删除，在这种新的情况中重新应用测试。
- 如果发生这样一种情形，即当我们试图去交换Q时却发现它已经被标记为删除，那么P和Q是相交的。在这种情况中，多边形Q可由多边形P的平面分割成两个部分，而且由Q分解出的两个新的多边形将插入列表中。

一旦通过这种方式渲染了多边形，就从列表中将它删除，并从列表的后部继续处理。当所有的多边形都经过了处理，场景就得到了渲染，所有隐藏表面都得到了删除。

在对象空间中排序多边形

现在让我们来看看在世界坐标中的多边形排序方法。不过在这之前，我们还是先给出两个多边形之间可见性关系的一个比较精确的定义。我们说从某个给定的COP位置C观察时多

边形P_1遮挡了P_2，当且仅当存在一条从C点出发的光线与P_1和P_2两者都相交，有$t_1<t_2$，如图11-4所示。如果不能找到这样的光线，那么两多边形可以以任意顺序渲染。

现在让我们设想有两个对象O_1和O_2，以及将两个对象分开的平面H_1，如图11-5所示。如果COP在H_1的正面，那么很容易证明不会存在一条光线是从COP出发，且在与O_1相交之前相交于O_2，这种光线会横穿H_1两次的。因此我们能够安全地在渲染对象O_1之前渲染对象O_2。COP位于H_1的反面的情况，可以进行类似的讨论。

这个思想源于Schumacker et al.（1969），可以扩展到场景中有超过两个对象的情况，如
248 图11-6中所示。通过测试 COP相对于H_1的位置，可以决定平面所分开的两组对象的顺序，但是在这两组里面我们将如何决定对象之间的顺序呢?我们可以继续放置越来越多的分割平面，直到在所定义的一组区域中每个区域只有一个对象为止。这种分割可以用一棵树结构来表示，其内部节点储存的是分割平面，而叶节点存储的是对象。该树在预处理阶段一次构造完成。因而可以通过在每一层上测试COP相对于平面的位置来确定顺序，先显示远端的对象再显示近处的对象。

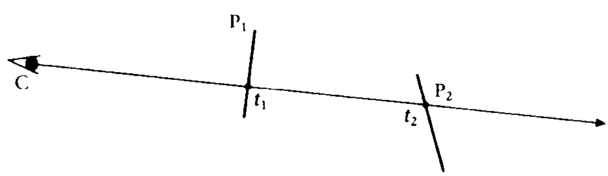


图11-4 多边形P_1遮挡了多边形P_2

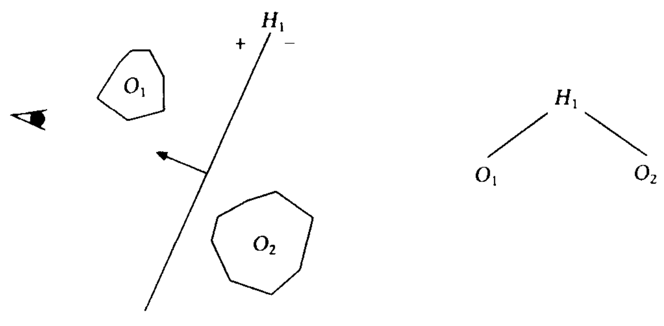


图11-5 使用分割平面决定对象间的顺序

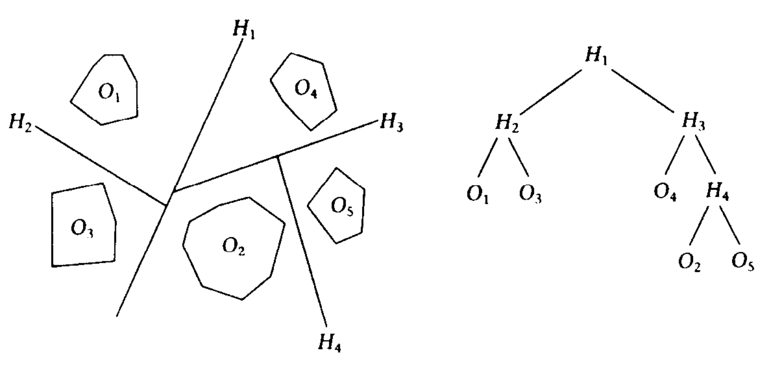


图11-6 一组对象和对应的树

在最初的实现中多边形是通过手工分成若干簇，分别对应于我们这里所说的对象，它们将被保存在叶节点中。对于在每簇中的多边形，其不变的排序是预先给定的。对于凸的或相当简单的对象来说，如果我们考虑背面删除，便能求出这样的顺序，如同先前讨论的一样。

构造这样一棵树的困难之一在于找到适当的分割平面。事实上很容易给出一种情形，不可能找到任何平面能将所有对象分割开来。同样，这在初始的实现中是由手工完成的。

像上述的算法在20世纪80年代中的飞行仿真和其他类似系统中占有重要地位，那时快速的硬件z缓冲区技术还没有得到广泛应用。然而，如果不是用于可见表面确定，而是用于如碰 249
撞检测（第20章）和可见性剔除（第23章）等高层操作，它们仍然将是相当重要的。

11.4 二叉空间分割树

二叉空间分割（binary space partitioning, BSP）算法是在几年后由Fuchs等（Fuchs et al., 1980; Fuchs et al., 1983）提出来的，作为对Schumacker分割树的一个简单而自然的扩充。它处理的是先前工作中关于不可再分的对象以及对象内排序中的每一个问题。

BSP树方法依赖于这样的一个事实，即多边形可以定义有“正面”和“背面”。我们在前面讲过，如果多边形的平面方程是 $l(x, y, z)=0$，这里 $l(x, y, z)=ax+by+cz-d$，那么它的正面法向是 (a, b, c)。对于任何点 (X, Y, Z)，如果 $l(X, Y, Z)>0$，那么该点在平面的“正面”；如果 $l(X, Y, Z)<0$，那么该点是在平面的“背面”（或“内部”）；而当 $l(X, Y, Z)=0$ 时，表示该点在平面上。

处理可见面确定问题的BSP树算法有两部分：首先是从场景多边形集合出发构造出树，然后假设一个视点，对树进行遍历以便得到所需要的从后向前的顺序。

BSP 树构造

一棵BSP树表示空间的一个递归细分。通常以某种方法选择一个平面，在场景中的所有多边形根据这个平面而分为在前面、在后面和在平面上三种类型。那些在平面上的多边形由该平面所确定，在平面前面的空间以及其中的多边形，以相同的方式继续递归细分，对于在平面后面的空间也同样处理。所选择的用来分割不同空间的平面称作**分割平面**。我们比较倾向于将分割平面选择为包含场景中的多边形的面，尤其对于在这一章中所考虑的可见性应用。根据上面的讨论，我们可以将BSP树的数据结构表示为：

```
typedef struct _bspTree{
  FaceList *face; /*list of faces belonging to this node*/
  PlaneEq plane; /*partitioning plane*/
  struct _bspTree *front; /*front node*/
  struct _bspTree *back; /*back node*/
} BSPTree;
```

假设一个由一组多边形定义的场景，其中一个多边形被选择作为根节点。所有其他多边形根据它们与根平面之间的位置关系被分入三个集合中——前集合（外侧）、后集合（内侧）以及根平面上。若是一个多边形的某些顶点在根平面的后面，一些顶点在根平面的前面，则将该多边形依根平面分解成两个多边形。树构造算法对前集合和后集合递归执行， 250
直到整个多边形集合处理完毕。这个过程是在对象空间中执行的。伪代码实现如程序11-1所示。

程序11-1　构造一棵BSP树

```
BSPTree *makeTree(face_list)
{
  if Empty(face_list) return EMPTY_TREE;
  else{
    root = select(face_list);
    on_list = NULL;
    back_list = NULL;
    front_list = NULL;

    for(each face in face_list){
      c = splitFace(root,face,front,back);
      switch(c) {
        case ON :appendList(on_list,face);
                 break;

        case IN_FRONT :appendList(front_list,face);
                 break;

        case AT_BACK :appendList(back_list,face);
                 break;

        case SPLIT :appendList(front_list,front);
                   appendList(back_list,back);
                 break;
      }
    }
    return Combine_tree(makeTree(front_list),
                 on_list,makeTree(back_list) );
  }
}
```

函数Combine_tree只是构造一棵树，其第一个参数是前节点，中间参数是根中的多边形的列表，而第三个参数是后节点。

Thibault 和 Naylor（1987）证明了也可以渐增地构造一个 BSP 树。一个多边形提供了根节点平面，然后后续的多边形沿着树过滤。对于一个给定的多边形，如果它完全在根平面的前面或后面，那么就将它（递归地）添加到根节点的相应子节点上，否则由根节点将它一分为二，然后再将每一部分递归地添加到对应的子节点中。显然在根平面上的多边形被增加到根节点。

在构造 BSP 树方面的主要计算任务是测试并在需要的时候把多边形分为两半。完成这一
251 步的算法包括搜索目标多边形的每个顶点p_0，p_1，…，p_{n-1}并对每个顶点计算$l(p_i)$。当l的符号发生改变的时候，这就意味着多边形穿越过了分割平面。如果 $l(p_i)=0$（对所有的 i），那么多边形是嵌在平面中的。这由程序11-2给出。

程序11-2　根据平面对多边形分类

```
/*evaluate plane equation at point p=(x,y,z)*/
float l(p, (a,b,c,d)) = return a*p.x + b*p.y + c*p.z - d;

void splitFace([p0 ,p1 ,...pn-1 ], (a,b,c,d), front, back, on)
/*splits the face by the plane equation*/
{
  front = back = on = NULL;

  /*find a first vertex of face that is not on the plane*/
```

```
j = -1;
for(i=0;i<n;++i){
  L0 = l(pi );
  if(L0 ≠ 0.0) {j = i; break;}
}
if(j == -1) {/*face on the plane*/
  on = face;
  return;
}

/*if reached here then the face isn't on the plane*/
if(L0 > 0) {/*front side of plane*/
  addVertex(front,pj );
  currentFace = front; otherFace = back;
}
else{/*back side of plane*/
  addVertex(back,pj );
  currentFace = back; otherFace = front;
}

/*assume that vertices are stored cyclically, ie, pn+k == pk*/
for(i=j+1; i < n+j; ++i){
  L1 = l(pi );
  if (sign(L0 ) == sign(L1 )) addVertex(currentFace,pi );
  else{/*change of sign*/
    p = intersection(pi-1 ,pi ,plane);
    addVertex(currentFace,p);
    addVertex(otherFace,p); addVertex(otherFace,pi );
    swap(currentFace,otherFace);
  }
  L0 = L1 ;
}
if(L1 >0) {
  front = currentFace;
  back = otherFace;
}
else {
  front = otherFace;
  back = currentFace;
}
}
```

252

BSP树场景的渲染

相对于任意照相机位置，对树中多边形进行渲染就是以一个特殊的顺序遍历这棵树。这基于Schumacker的核心思想，即与视点位于平面相同一侧的对象不会受到另一侧对象的遮挡。尤其是对某个树节点上的平面，相对于 COP来说远的那一侧应该先渲染，然后渲染在节点上的多边形，最后是对节点近端的子树进行渲染。这是精致而简单的树遍历算法，其复杂度是树节点数的线性函数。

COP（在WC中）由根多边形平面分为几种类型。如果它在根节点的前面，那么遍历算法首先递归地处理根节点的后集合，然后显示根节点多边形，最后再遍历前集合。如果它在根节点的后面，那么树的遍历要先处理前集合，然后显示根节点，最后处理后集合。注意，显示函数负责裁剪、投影和渲染。如程序11-3所示。Gordon和Chen（1991）已经证明，对于高深度复杂度的场景，对渲染速度的一种重要改善方法是将树的由前至后的遍历和扫描线多边

形填充的动态数据结构结合起来使用。

程序11-3　根据COP遍历BSP，产生一个由后向前的列表

```
void TraverseTree(tree,COP) {
  if(EMPTY(tree)) return;
  else{
    if(COP in front of rootPolygonOf(tree)){
      TraverseTree(BackDescendent(tree));
      Display(rootPolygonOf(tree));
      TraverseTree(FrontDescendent(tree));
    }
    else{
      TraverseTree(FrontDescendent(tree));
      Display(rootPolygonOf(tree));
      TraverseTree(BackDescendent(tree));
    }
  }
}
```

由Fuchs 等（1983）给出的 BSP 树实现说明无需特殊的图形硬件，这个方法对于它们的模型来说能得到接近实时的帧频率，虽然树本身的计算时间可能是很大的。BSP 树因此特别
253 适合于场景本身是静态的而照相机时常移动的一类应用。举例来说，建筑漫游就是这样一种情况，照相机在一群建筑物当中不断移动（一个较早的例子是Brooks，1986），类似应用还有很多。

构造一棵比较好的树

这一章中我们专注于用 BSP树解决可见性排序问题，其实 BSP 树是一种有效的工具，能解决许多其他的问题——碰撞检测、视景体选取、可见性选取等等（Naylor，1993）。构造一棵树适合于所有的操作是件非常困难的事情，因为不同的操作会有不同的需求。这种树有两个属性需要调节，即规模和形状，它们对于不同应用的意义是不一样的。

如 Paterson and Yao（1990）以及Yao and Paterson（1989）所示，对于初始多边形数n、构造一棵 BSP 树的时间复杂度和空间复杂度的上界都是$O(n^2)$，虽然理想情形复杂度接近$O(n \log n)$。时空复杂度会由于在每次递归中作为根节点的分割多边形的选取不同而有很大差异。

控制树的规模和形状的常用方法是在每一个递归过程给出几个候选多边形（也许是 5个或10个），并找出最好的一个作为根节点（Fuchs et al.，1983）。对每一个的评估是通过在子空间中将它与其余多边形比较，并且计算两个量的加权和，这两个量分别是规模（产生的分割数）和分布（在每个所得到的子集合中多边形数目的差别）。

所使用的权值依赖于应用。对于可见表面确定，在每一帧中都要访问树的每个节点，不需要搜索。因此树的平衡是无关紧要的。节点数目和多边形数目是重要的。如果有太多的多边形被拆分开的话，就会发生问题，因为这时多边形数据库规模将急剧增大，由此渲染过程相应变慢。另一方面，对于光线跟踪应用或是包括分类的算法，树的平衡将比规模更重要。同时，平衡树通常能够比较快速地构造出来（如果拆分的数目不是非常大的话），但这不会影响到运行时的性能。

另一个思想是首先由Slater（1992a）提出来的。这里，多边形在树的构造过程中出现顺序是一种从“周围”到“中心”的顺序——那些在场景的外围的多边形被用来作为靠近中心

的多边形的根。这适合于描述建筑物内部的场景——所以比较大的处于周边的多边形（比如表示墙壁的多边形）不太可能拆分那些更靠近中央的对象（比如房间里的家具）。

Naylor（1993）提出了一种不同的优势量度，它基于概率模型表示的各种操作的期望代价。简单地讲，其思想是让较大的单元（更可能被访问到的）位于较短的路径上，让较小的单元位于较长的路径上。从某种意义上说，这是类似于包围体的一系列逼近。

当BSP 树用于可见表面确定的时候，需要执行视点与树的每个分割平面的点乘运算。这会是一种代价很高的运算。近几年来，人们提出了很多 BSP 树的变化形式，努力降低这个代价（Chen and Wang，1996；James and Day，1998； Sadagic and Slater，2000）。在这些方法中总是牺牲一些方面来换取速度的提高。这些牺牲通常是内存耗费和树的适用性（比如它们当中没有一种可以用来做树的合并）。 254

在动态场景中使用BSP 树

BSP 树表示对于场景几何性质保持不变的一类应用的确是很有用的，这类应用主要关心的是在场景中漫游，因为照相机位置的变化只需要在一个不同的顺序中遍历树。当场景的几何性质发生了改变，最初的 BSP 树将不再是一个有效的表示，因为定义分割节点的平面可能已经改变。然而，我们可以构造算法来修补由于几何变换或对象删除所带来的损失。

在Schumacker 的基于对象的分割树中，内部节点由分割平面定义，而不是用多边形平面来定义，对象能自由地移动而不会引起任何问题，只要它们不越过任何分割平面。Torres（1990）在稍后的时间里提出了一种相似的思想。在他的这个算法中，每个在叶节点上的对象也被表示为一棵BSP 树，这棵BSP 树是根据对象自己的多边形构造出来的。如果一个对象移动，只要不穿越分割平面，它的树就保持有效——仅仅可以被平移。

Fuchs 等（1983）提议说，如果我们预先知道有哪些动态对象以及它们的路径，就有可能对它们采取一些有效的处理方法。可以构造一棵树，让运动区域封闭在一个树单元里面。那么对象能独立地在该区域中移动，树中其余部分不会受到影响。

Naylor（1990）提出了一种不同的方法，该方法也是需要预先知道将有哪些运动对象，但不使用它们的路径。这或许是对于动态变化场景的一种最优美的方法了；然而，它利用了树合并（ Thibault and Naylor，1987），这已经超出了本书的范围。简要地讲，首先将静态对象放在一棵树内，将每个动态对象分别放入单独的一棵树中。在每一帧的开始时刻，我们可以将所有的树合并在一起，这样得到一个完整的场景树。利用一些树的复制我们可以在后续的帧中使用单个树。Naylor 描述了一个这样的应用，其中用户可以交互地向工件添加一个配件或从工件中减去某个配件。

Chrysanthou和Slater（1992）提出了一种不需要对移动对象有预备了解而又非常容易实现的方法，尽管它的效率不如Naylor的方法。在这个小节其余部分中我们将会更详细地讨论这个方法。 255

在虚拟环境中通常会是只有一个非常小的部分场景发生了改变，举例来说用户走向一个对象并将它拾起，而其余部分场景保持不变。在这种情况下，我们可以利用这个简单方法。每当变换一个对象，无论是对它进行平移、旋转还是伸缩，甚至是变形，都要反映到树中。首先删除对象的多边形，然后对对象的几何属性（多边形的顶点）进行变换，然后再将这些变换后的对象多边形添加到树中。通过对BSP树的渐增式构造，很容易看到多边形是如何被添加回去

的；它们可以像一般情况一样沿着树过滤。这个算法的主要问题是该如何删除多边形。

从树中删除一个多边形似乎是一个困难的操作，因为由多边形定义的平面可能构成一个节点，而这个节点将进一步拆分子空间。此时删除一个多边形和它的节点会将树拆分成两个部分。然而，经过进一步的细致分析，我们发现有四种情况——被删除的多边形可能是下列情况：

（1）在一个叶节点上：此时多边形的删除是容易的。这个删除是有效的，因为由多边形定义的平面拆分了一个“空的”子空间。

（2）与其他面共同位于一个节点上（即在相同的平面上）：此时多边形可以从这个节点删除，因为平面仍然由那个节点上的另一个多边形所定义。然而，如果在这个节点上的前两个多边形面向相反的方向，我们删除了第一个面，那么该节点的前后子树必须交换，以便保持树正确的前后顺序。

（3）节点恰好只有一个非空孩子：此时，该平面将子空间拆分成一个空的区域和一个非空的区域。所以如果删除了这个节点，它将由代表非空区域的那个节点所代替。换句话说，被删除节点的孩子将直接变成这个节点的父亲的孩子。

（4）节点有两个非空的孩子：这是最困难的情况。多边形拆分子空间为两个非空的区域。所以如果删除了多边形，我们会得到两个不相连的子树。可以将它们重新构造到一棵BSP 树中，方法是沿着最大的树向下过滤较小的那棵子树中的多边形，可以使用前面讨论过的渐增BSP 树的构造算法。Torres也用到了这个方法，不过他是在对象之间无法找到一个分割平面的时候使用它。

这四种情况可以通过图11-7中的场景来说明，其相应的树由图11-8给出。情况1一般用于
256 删除多边形4或7的情形中。情况2一般用于删除多边形3或6的情形中。注意，如果3被删除，那么两棵子树一定要交换过来。情况3一般用于多边形1和5的情形，情况4一般用于当2被删除的情形，这种情况所对应的树如图11-7所示。

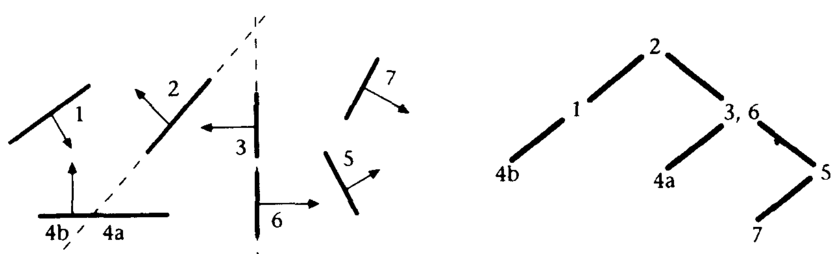


图11-7 一组多边形及其BSP树的二维表示

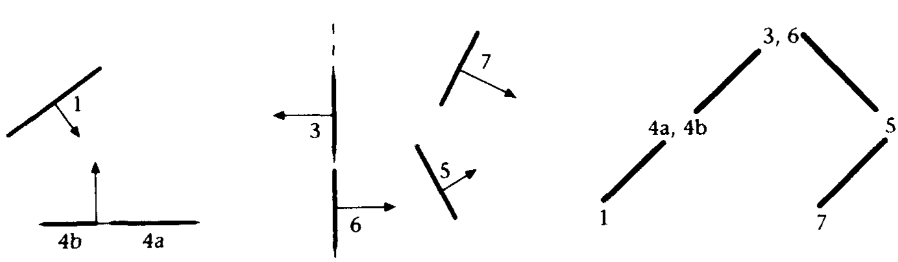


图11-8 将多边形2删除后的树

算法的运行分为两个阶段。对于移动对象中的每个多边形，当多边形是节点上惟一的一个时，将该节点标记为“删除”。这可以通过两种方法完成。一种方法是搜索多边形，另一种更好的方法是跟踪每个多边形在插入树中时的储存位置指针。当在节点上有超过一个多边形时（情况2）就立刻删除该多边形，无需再对节点做标记。如果它是第一个多边形，第二个多边形朝向相反的方向，那么对那个节点上的前后子树进行交换。

在所有有关节点做过标记之后，对每个节点调用一次递归函数（Restore），删除有记号的那些节点。函数Restore在程序11-4a中给出。简而言之，在每个迭代过程中对根进行检查。如果它没有标记，函数转而去处理它的左子树和右子树。如果根是有标记的，而它只有一棵非空子树，算法将处理子树然后返回。如果两棵子树都是非空的，那么它将求出两者中最大的那个，恢复它并返回一棵树，这棵树是将比较小的那棵树的多边形插入所得的新树（FilterIntoTree）。

程序11-4a 恢复BSP树来考虑被删除的多边形

```
BSPTree Restore(tree)
{
   if(Empty(tree)) return NULL;
   if(tree.root is not marked as "deleted") {
     tree.front = Restore(tree.front);
     tree.back = Restore(tree.back);
     return(tree);
   }
   else{
     if(any child of the tree is empty) return(Restore(other child
     of tree))
     else
     return( FilterIntoTree(polygons of smaller sub-tree,
     Restore(largest subtree) ) );
   }
}
```

257

使这个算法比较实用的因素之一是，当目标对象被变换的时候，比如在一个交互式应用中，Restore函数只相关于最初的变换。理由是在对象多边形从树中删除之后，经过对它们的转换并重新插入，它们将结束于三个地方：在树的叶节点上；在一个共享节点上，该节点有同一平面上的其他多边形；在靠近树的叶节点的一个节点上，即在包含目标对象全部多边形的子树中。在这样一个特别的交互序列中，每种情况下只有比较简单的删除情况会用在这个对象的后续变换中。

其次，算法的结果是那些多边形靠近 BSP 树的叶节点的对象基本上可以在常数时间内被删除。因此对于那些需要经常变换的对象，比如在交互式应用中的一个3D光标对象，或者是房间内部比较小的对象（如房间的摆设等），应该被最后放入树中。

第三是程序11-4a中函数Restore的方法，当要删除的节点拥有两个子节点时，较小的那棵子树要渗透到较大的那棵子树中，无论两棵树的规模有多大。这可能是一个耗费很高的操作，而且可能是很浪费的，因为在合并后的子树中多边形可能只有一个很短的生命期。一个替代策略是采用一些准则来确定什么时候渗透操作应该执行，或者什么时候节点仅仅被标记为“删除”而没有真正从树中删除。所采用的准则只有在较小的子树小于某个最大规模的时候才做渗透操作。此时，Restore函数应该改变，这样第二个“if”语句如程序11-4b所示。

程序11-4b 算法1(a)的改进

```
if(ree.root is not marked as "deleted" || the smaller subtree is
too large){
   tree.front = Restore(tree.front);
   tree.back = Restore(tree.back);
   return(tree);
}
258 else{/*as before*/
}
```

应该注意的是，在一个场景中可能只有非常小比例的多边形造成树构造中绝大多数的分割现象。正是这些多边形将有大的子树。因此在树中保留标记为“删除”的多边形的操作不会经常发生，也因此不会过度地增加树的规模。同时，如果这些标记为“删除”的多边形保留在树中，对其他对象进一步的变换可能减少标记为“删除”的多边形的子树规模，所以它们无论如何最后是要被删除的。

11.5 小结

在这一章中我们介绍了列表优先权算法，它根据相对于COP点的可见性来排序多边形。对于“大场景”，没有了光线跟踪所带来的可见性“证件”，这个算法就成为我们为构造新系统“找回”多边形可见性的第一个有力工具。事实上，现在几乎有了一个完全新的系统：我们用多边形作为基本元素来描述图形对象，以对象的层次结构来描述场景。我们能从任意视点并以正确的可见性顺序来渲染场景。但这仍然存在问题。

首先我们注意到，尽管BSP树方法对于可见性是有效且快速的，但同时也是需要很大内存的。如果在构造过程中不精心设计的话，数据结构的规模将以多边形数目的二次方的速度增长，因此对于具有数以百万计多边形的场景来说是相当困难的。然而一个更明显的问题是在BSP树构造中的分割过程中：会产生越来越小的多边形碎片，导致不精确性在逐渐放大。第二个问题是我们还没有重新把光照导入场景。如何对显示的多边形进行明暗处理呢？第6章所介绍的光照计算是怎样引入场景中？第三，BSP树方法结束了二维多边形在视平面上渲染的需要。二维多边形是如何渲染的呢？所有这些问题——不断增长的内存需求、重新导入明暗处理和多边形的渲染——都存在一个解决方案，而这个解决方案与二维多边形的渲染有密
259 切关系。

在下一章中我们将会介绍二维多边形渲染算法。还会在下下一章中说明如何使用该算法于另外一个解决可见性问题的方法，叫做z缓冲区。我们还将使用相同的思想说明如何轻松地将明暗处理加入到渲染过程中。

第12章 多边形渲染

12.1 引言

BSP树方法是优先权列表算法的一个例子。当按照一个特定的照相机设置进行遍历的时候，它预先计算一个数据结构，这个数据结构能产生场景多边形的排序。理论上讲，这个数据结构的计算是用世界坐标表达的场景进行的，也就是，它是独立于任何特殊照相机的。这是一个视图独立的方法。为了得到正确的视觉效果它可以依赖光栅显示器的过着色性质——以从后向前顺序渲染多边形将会得到正确的可见性的场景可视化效果。

如我们已经看到的，投影之后多边形可以看成是在XY投影平面上的二维对象。优先权列表算法预先排序多边形，这样就能保持正确的深度顺序。因此假若我们以一个正确的顺序渲染，多边形的z深度就不再重要了。因此，BSP 算法产生的由后向前序列只需要渲染二维多边形序列。

这又提出了一个问题——如何高效地进行二维多边形的渲染?这个问题正是本章所要讲的内容。我们将会在下一章中看到所得到的算法还有一些令人惊奇的扩充算法——它可以用来提供可见性问题的完全不同的解决办法，同时能够用来提高明暗处理效果。然而此刻我们将专注于二维多边形渲染算法本身。 260

12.2 多边形光栅化

在多边形内部

背面删除在渲染管道中是重要的第一步。多边形的光栅化，即找出最佳的像素集合来表示一个多边形，（几乎）是最后的一步。WC对平面多边形的标记方法与第8章所讨论的一样。我们假设顶点已经转换成像素位置，也就是说已经用整数坐标空间表示了。

我们曾经定义过，多边形是简单的，如果它没有两条边存在交叉（除了在顶点处相交外）。多边形是凸的，如果它是简单的而且每个内角$\Delta p_{i-1} p_i p_{i+1} < \pi$。否则我们称多边形是任意的或复杂的。（一个简单多边形是所谓的约旦曲线的一个特例，也就是说，任何在拓扑结构上等同于圆的曲线。）

为了要填充一个多边形，找出多边形边界上的所有像素点和内部像素点，必须对多边形的“内部”和“外部”给出明确的定义。如果多边形不是复杂多边形，那么每个人对什么是（多边形的）内部的认识就不会有什么差别（虽然在不同的环境中我们可能需要考虑是否将边看作内部或外部）。如果多边形是复杂的，那么对内部或外部就没有一个“确切”的答案了。然而，对于特别的应用来说，我们可以给出规则，定义什么是所希望的内部和外部。

有两个常用规则如下所示:

奇偶规则。对于任意一点，从这一点作一条无限长的水平直线。计算这条直线与多边形边的交点数目。如果交点数是奇数，那么该点在多边形的内部。

如果直线穿过多边形的一个顶点，可以对直线作扰动处理使得这种情况不再发生，否则规则就无法奏效。或者将最大顶点或最小顶点计数为0，所有的其他顶点计数为1。边上的点或顶点（显然在边上）的分类与所在边本身的分类相同。

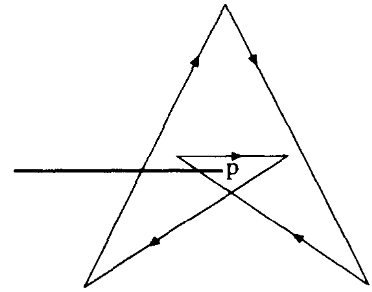


图12-1　非零匝数规则

非零匝数规则　我们假设对边按照某一个确定顺序遍历，比如按顺时针遍历或按反时针遍历。图12-1给出了一个
261 顺时针方向的例子。像前面一样，从任意点画一条无限长的水平线。计算在穿越直线的这些边中方向朝上的边的数目u和方向朝下的边的数目d。如果$u-d=0$，那么点在外部，否则它在内部。（其物理意义是计算多边形绕相关点的次数。）

利用相关性

填充一个多边形需要找到所有位于多边形内部的像素，并将它们设置成所需的颜色。这可以用一种粗糙的做法完成——对每个像素执行一次是否在多边形内部的测试。作为一个串行算法这会是非常低效率的，更不适于在并行机上计算。这种做法忽视了*像素相关性*——也就是说，如果一个像素是在内部或外部，那么它周围的像素一般也和该像素具有一样的性质。

比较好的方法是找出多边形最上面的顶点和最底下的顶点，分别具有y坐标值ymin和ymax。然后对于每一条在ymin和ymax之间的扫描线计算它与多边形边的交，并用适当的颜色填充这些水平区间。这样做是可以的，但是效率仍然不高，因为它没有充分考虑*扫描线相关性*——任何一条给定的扫描线的交点的x坐标值与其上方和下方的扫描线的交点x值之间是有关联的。一个高效率的算法会充分利用这些信息，如图12-2所示。

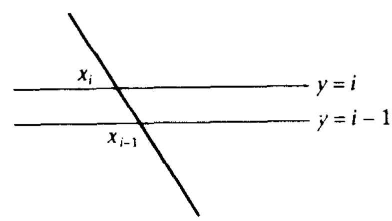


图12-2　扫描线相关性。一条扫描线的x交点值可以从先前的扫描线计算出来

让我们看一下两条连续扫描线与一条边相交的交点。边的方程为$y = a+bx$，这里b是直线的斜率（dy/dx）。我们假设$b \neq 0$。

$$\begin{aligned} i &= a+bx_i \\ i-1 &= a+bx_{i-1} \end{aligned} \qquad (12\text{-}1)$$

因此，

$$x_i = x_{i-1} + \frac{1}{b} \qquad (12\text{-}2)$$

边表

下面的算法利用扫描线的相关性，并基与Appel（1968）、Bouknight and Kelley（1970）
262 以及Watkins（1970）的方法。考虑从点（x1，y1）到（x2，y2）的一条边，边的标记顺序是按照第一个点的y值总是小于第二个的y值（y1<y2）。水平边我们全都忽略掉。这个边表达为如下结构：

```
struct Edge {int y2; float x1; float Dx;}
with Dx = dx/dy
where dx = x2-x1, dy = y2-y1.
```

所有的边根据它们较小的y值（y1）做桶式分类。这个分类序列称为边表（ET）。因此边表包含了最初多边形中所有的边，但是现在依照它们开始的高度排序（即按照比较低的y坐标值排序）。因此，与ET内的每个桶相关联的将是一组边，这些边有一个相同的y坐标值（较低的那个），该坐标值等于桶的指针。

更明确地，我们可以将ET表示为一个数组，其边界值分别是0和在显示屏幕上最大可能的扫描线的y值（YMAX）。每一个数组入口是一个三元组序列，每一个三元组表示的是一条边，边使用的是上面的边结构。为了要构造这个边表，

```
Initialize ET[i] = Ø(the empty sequence), for i = 0 to YMAX;
```

对于每条从（x1，y1）到（x2，y2）的边，保证有y2> y1（可能需要进行值的交换，并忽略掉水平边）：

```
append(ET[y1],{y2,x1,Dx})
```

该函数将新的三元组放于这个列表的尾部（也可以将它放在列表的头部，或根据x1值顺序插入到列表的相应位置）。在构造ET表时，跟踪顶点y坐标的最大值和最小值（ymin和ymax）。

接下来要处理ET表，处理过程如下所示。我们要构造一个新对象AET（活动边表），它是“sequence of Edges”类型（即与任何个别的ET[i]类型相同）。

```
processET(void)
{
  AET = Ø;                          /*initialise to the empty sequence*/
  for(i = ymin; i <= ymax){         /*for each scan-line*/
                                    /*delete from AET entries with y==i,*/
    update(AET,i);                  /*and compute x1 += Dx for remainder*/
    append(AET,ET[i]);              /*join ET[i] to the AET*/
    sort(AET);                      /*sort the entries by x1*/
    JoinLines(AET,i);               /*join horizontal lines between*/
                                    /*pairs of x1, at height i*/
  }
}
```

活动边表是一个边的序列，其中的每一条边与当前扫描线相交（多边形边和扫描线的交），这里当前扫描线是按照x1值渐增的顺序排序的。最初活动边表为空——任何位于多边形最低顶点（在y=ymin）下面的扫描线显然不与任何边相交。函数Update删除在AET中的那些与位于i高度的当前扫描线具有相同y坐标的边（因为这些边现在已经被完全处理过了，而且位于当前扫描线的下面）。对于那些未被删除的边，用增量Dx加到它们的x1坐标上，实现图12-2中所示的扫描线相关性思想。这些x1坐标是活动边与当前扫描线交点的x坐标。函数Append将那些开始于高度i的边添加到 AET 中。函数sort在AET中依照边的x1值大小对边进行排序。其原因是下个函数JoinLines必须将成对的x1值所确定的水平区间连接在一起。这要靠奇偶内部检测方法保证正确性（假设当 AET为空的时候，更新没有效果）。 263

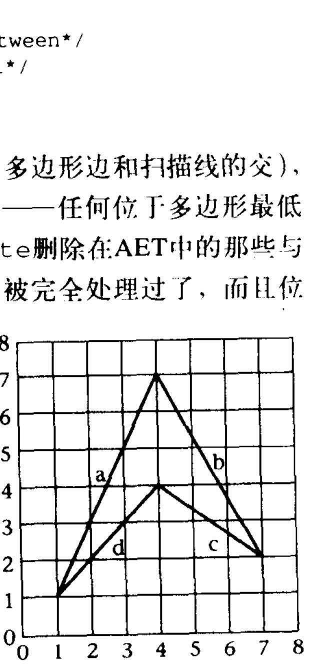


图12-3 一个多边形例子

图12-3 给出了这个思想的一个多边形例子。多边形有边a、b、c和d，如表12-1所示。

这个图所对应的边表在表12-2中给出，其AET表在表12-3中给出。

表12-1　图12-3中多边形的边

边标号	坐标	$y1$	结构
a	(1,1)到(4,7)	1	(7,1, 3/6)
b	(7,2)到(4,7)	2	(7,7, − 3/5)
c	(7,2)到(4,4)	2	(4,7, − 3/2)
d	(1,1)到(4,4)	1	(4,1, 3/3)

264

表12-2　图12-3的边表

$y1$	边序列
1	(7,1, 3/6), (4,1, 3/3)
2	(7,1, − 3/5), (4,7, − 3/2)

表12-3　图12-3的活动边表

扫描线i	活动边表	高度i处的跨距
0	空序列	
1	(7, 1, 0.5),(4, 1, 1)	1至1
2	(7, 1.5, 0.5),(4, 2, 1),(7,7, − 0.6),(4,7, − 1.5)	1.5至2, 7至7
3	(7, 2.0,0.5),(4, 3, 1),(4.5.5, − 1.5),(7, 6.4, − 0.6)	2.0至3, 5.5至6.4
4	(7, 2.5,0.5),(7, 5.8, − 0.6)	2.5至5.8
5	(7, 3.0,0.5),(7, 5.2, − 0.6)	3.0至5.2
6	(7, 3.5, 0.5),(7.4.6, − 0.6)	3.5至4.6
7	空	
8		

注意x1和因此而得到的水平跨距是浮点数，这对于表示精确的交点是理想的。然而，跨距必须对应精确的像素位置，因此小数值是走样为什么发生的另一个例证——这次是沿着显示多边形的边。

很容易说明（实际上已经在例子中给出了）这个算法不能显示最大的顶点，也不能显示水平边。这一点很容易修补，可以有多种方式——举例来说，直接渲染这些丢失的元素或对于最大顶点和水平边这种情况，做增量x1 + = Dx并在删除这些实体之前画这些跨距。然而从另外一个角度看，这种缺陷可能也是想要的。

在三维应用中，经常有这样一种情况，有大量的多边形需要渲染，这些多边形之间通过边彼此连接在一起，如图12-4所示。为了渲染，这些三维多边形被转换成显示屏幕上的二维多边形（通过投影过程），如我们所看到的那样。显然一些边邻接多边形之间的公共边。在这种情况下，我们可以采用一些规则，如公共边只处理一次。经常采用的一条规则是扫描线上最右侧和最上侧的像素不渲染，因为我们知道当前多边形是与右侧或上侧的那个多边形相连接的。这样这些边将不会被渲染两次。

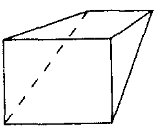


图12-4　一个三维对象是由连续的多边形构成的

265　12.3　小结

本章的核心是二维多边形填充算法。多边形填充无疑可以找到非常高效的实现方式——

尤其是当初始多边形经过进一步处理变成三角形之后。这样甚至没有必要保存边表了（因为每条扫描线每个多边形只有两个交点，不存在极小和极大），但是扫描线相关性显然是需要维护的。有一个相关的渲染多边形的替代算法，它是基于对并行硬件处理研究的，可以参见Pineda（1988）。

在下一章中我们将看到如何使用二维多边形算法来解决深度计算问题和平滑明暗处理问 266
题。这样我们将完成完整的3D图形管道的分析和介绍。

第13章 图像空间渲染和纹理生成

13.1 引言

这一章我们转向图像空间算法。这里可见性计算被放在“最后”——也就是在像素层进行。在光线跟踪的情形中，我们通过一个像素跟踪一条光线，并在所有的对象之中搜寻以找到该光线所交的最近的对象（如果有的话）。正如我们所了解的，这是一个很慢的过程。这里我们要讨论一种相反的算法：将每个多边形投影到视平面上（以任何顺序）。设定每个像素的颜色，颜色的设定是根据从该像素所能“看到” 的多边形确定的，这个多边形对该像素来说具有最小的深度值（z）。在完成对所有场景多边形的投影后，像素点将只显示从那个像素可以看到的多边形的颜色（或为背景的颜色）。这有如光线投射的效果，但是要比光线投射快几个数量级。使用光线投射时，对于每条光线我们搜寻所有对象以便找到最近的相交点。采用
267 这种图像空间的方法，我们投影每个多边形，对它所覆盖的每个像素，我们只根据所有覆盖这个像素的多边形中“较近的”（有较小的z深度值）的那个多边形来设定像素的颜色。因此，通过图像空间方法，总的渲染时间与场景中多边形数成正比。

图像空间方法提出了这样一些问题：

- 我们如何能高效地确定由一个多边形覆盖的像素集合？
- 我们怎样维护每个像素所需的深度信息？
- 我们如何确定像素应该被设定的颜色？

令人惊讶的是，对于这三个问题我们能找到一个统一的答案——这就是在第12章中所介绍过的二维多边形渲染算法。这个算法也被称作“3D图形管道的核心算法”——但是它本质上实际是一个二维空间中的算法。

在这一章中我们将要描述三个图像空间可见性算法。在此之后，将说明二维多边形渲染算法如何也能够用于明暗处理。我们始终假设我们已经执行了裁剪过程，而且已经将场景转换到了投影空间，这在相当大程度上简化了计算。最后，我们详细地讨论渲染当中一个最为重要的方面，一种能在相对低廉计算代价下获得图像逼真外观的方法：纹理映射的思想。

13.2 z缓冲区可见性算法

基本思想

我们在最著名的图像空间可见性算法（即z缓冲区算法）的高效实现中使用了二维多边形渲染算法（由Catmull给出，1974）。这是基于帧缓存关联数组的思想（像素对数组元素的一对一关联），这个帧缓存在执行算法期间，保持像素相对应的“深度”信息，像素至此有了它们自己的颜色设定。假设显示分辨率是 $M \times N$。设z是与分辨率维数一样的一个数组，每个元素的初值设定为 1.0（在PS中最大的z值）。算法对场景中每个对象的处理过程如示：

（1）对象表面上的每个点（x，y，z）对应于显示屏幕上的一个像素（x_s，y_s）（经过平移

和缩放变换从 PS 转换到了显示坐标)。设 $I(x, y, z)$ 是在场景的适当明暗处理模型中这个点的颜色强度。

(2) 如果 $z < Z[x_s, y_s]$,那么设置像素 (x_s, y_s) 的强度为 $I(x, y, z)$,并将 z 值赋值给 $Z[x_s,$ 268
$y_s]$;否则什么也不做。

因此对于现在正在处理的像素,它的颜色如果需要重写,那一定是新值所代表的场景中的点较之先前颜色所代表的场景点距离COP 更近。z缓冲区的角色是记录在帧缓冲区中当前设定的颜色强度所对应的距离。

多边形扫描线渲染器的使用

在通常情况下,如果场景是用多面体表示的,那么每个多边形的处理可以通过扫描线使用在第12章中介绍的算法完成。对算法的明显改进必须检查每个单独像素,更新z缓冲区,而不是简单地用JoinLines子程序中的一条水平线来画当前跨距。

现在我们将分析对每个像素计算 z值的三种方法,从算法效率最低到最高,依次如下。

方法1:直接从平面方程计算。平面方程为 $ax+by+cz+d=0$。因此,给定点 (x, y),我们能求出 z 为:

$$z = \frac{d - ax - by}{c} \tag{13-1}$$

这是一个代价很高的计算,必须对每个像素执行一遍。代替算法是:

方法2:渐增计算。考虑在扫描线上两个连续的像素位置 y,(x, y, z_x),$(x+1, y, z_{x+1})$。将它们代入平面方程中,经过相减,我们得到:

$$z_{x+1} = z_x - \frac{a}{c} \tag{13-2}$$

对于像素在垂直方向上的遍历有类似的结果。

方法3:插值。我们记得在填充算法中,多边形的每一条边 (x_1, y_1) 到 (x_2, y_2),有 $y_2 > y_1$,表示为如下的数据结构:

```
{int y2, float x1, float Dx;}
where Dx = dx/dy;
```

扩充这个数据结构,使之包括 z 有:

```
{int y2, float x1, float Dx, float z1, float Dz;}
where Dz = dz/dy;
```

现在它表示一条边 (x_1, y_1, z_1) 到 (x_2, y_2, z_2),有 $y_2 > y_1$。在填充算法的更新阶段,对 z 的处理与对 x 的处理完全相同,即 z_1 被替换成 z_1+D_z,这里 D_z=dz/dy。这为从一条扫描线到下一条扫描线,沿多边形的边对 z 插值提供了条件。

除此之外,在多边形区域内水平插值是需要的。这是在多边形的填充算法在执行水平跨距渲染时完成的。假设在活动边表中对于扫描线高度 y 有两个连续入口,它们是 $(y_{a2}, x_{a1}, D_{ax}, D_{az})$ 和 $(y_{b2}, x_{b1}, D_{bx}, D_{bz})$。那么从 z_{a1} 到 z_{a2} 的插值,通过加如下量

$$D_{zx} = \frac{z_{b1} - z_{a1}}{x_{b1} - x_{a1}} = -\frac{a}{c} \tag{13-3}$$ 269

很明显,我们需要第一像素有 z 值 z_{a1},最后一个像素的 z 值为

$$z_{a1} + (x_{b1} - x_{a1}) \times D_{zx} = z_{b1}$$

这个过程称为双线性插值。当我们在下面考虑明暗处理的时候将会回过来继续讨论它。

z缓冲区多边形填充的例子

考虑在图13-1中的多边形和对应的边表。这个多边形存在于投影空间中，它的z值是给定的。（正常情况这个z值是在0和1之间，但是这里为了说明目的，我们使用的z值在这个范围之外。）我们要说明如何渐增地计算z值，这是填充多边形扫描线算法的一部分。

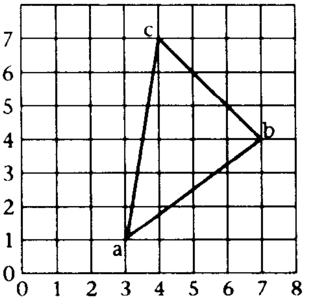


图13-1　多边形示例。a=(3,1,1)，b=(7,4,2)，c=(4,7,4)

ET入口的一般形式为：（$y2, x1, \frac{dx}{dy}, z1, \frac{dz}{dy}$），且

$$ET(4) = \left[cb \to \left(7, 7, -1, 2, \frac{2}{3}\right) \right]$$

$$ET(1) = \left[ac \to \left(7, 3, \frac{1}{6}, 1, \frac{3}{6}\right), ab \to \left(4, 3, \frac{4}{3}, 1, \frac{1}{3}\right) \right]$$

其他的$ET[i]=\varnothing$。

扫描线 y=1。

$$AET = \left[ac \to \left(7, 3, \frac{1}{6}, 1, \frac{3}{6}\right) \right], ab \to \left(4, 3, \frac{4}{3}, 1, \frac{1}{3}\right) \Big]$$

这对应于一个单一像素（3，1），这里z深度为1。

扫描线 y=2。

270

$$AET = \left[ac \to \left(7, 3.1667, \frac{1}{6}, 1.5, \frac{3}{6}\right), ab \to \left(4, 4.3333, \frac{4}{3}, 1.3333, \frac{1}{3}\right) \right]$$

这代表了在高度y=2上的一个跨距，从 x=3.16667 到 x=4.3333，即从像素（3，2）到像素（4，2）。对应的 z深度值为 1.5 和 1.3333。

扫描线 y=3。

$$AET = \left[ac \to \left(7, 3.3333, \frac{1}{6}, 2.0, \frac{3}{6}\right), ab \to \left(4, 5.6667, \frac{4}{3}, 1.6667, \frac{1}{3}\right) \right]$$

这代表了在高度y=3上的一个跨距，从x=3.3333到x=5.6667，即像素（3，3），（4，3），（5，3）和（6，3）。x的范围是从3到6，z的范围是从2.0到1.6667。因此，x增加一个单位造成

的变化为dz/dx= − 0.1111。因此每个新的z值可以通过把 dz/ dx 加到每个旧值上计算出来。举例来说：

- 当 x=3，z=2.0
- 当 x=4，z=2.0 − 0.1111=1.8889
- 当 x=5，z=1.8889 − 0.1111=1.7778
- 当x=6，z=1.7778 − 0.1111=1.6667

扫描线可见性算法

这是对上面所提到的标准二维多边形填充扫描线算法的一个更复杂的修改，因为它一次要处理场景中所有的多边形的边，而不是像z缓冲区方法那样一次只处理一个多边形。（这个方法已经有很长的历史了： Wylie et al.，1967; Bouknight，1970; Bouknight and Kelley，1970; Watkins，1970。）下列讨论假设多边形彼此不相交（除了在公共边之外）。渲染一个密集遮蔽的多边形需要两个主要步骤：首先，构造一个表示多边形边的数据结构（边表，ET）；其次，对这个边表进行处理（构造动态的活动边表，AET）。在第一个步骤中对多边形的每条边做遍历，相应的记录（y_2，x_1，$\frac{dx}{dy}$）添加到边表的元素y_1处（对于边（x_1，y_1）到（x_2，y_2），有$y_2>y_1$）。该算法的3D版本是相似的，多边形的每条边都表示在ET表中，除了在ET表的构造中忽略了z坐标。每一条边是一条记录，包含四个元素（y_2，x_1，$\frac{dx}{dy}$，pt），这里pt是“多边形指针”。pt项指向边所属多边形的信息。特别是，它的平面方程系数（a，b，c，d）通常需要储存起来，连同明暗处理所需要的任何其他信息（例如颜色系数）。

AET 的处理与二维的情况类似，如在子程序JoinLines的解释中所看到的，它负责将扫描线上连续的水平线段连接在一起。此时所发生的事情可以结合图13-2进行讨论。 271

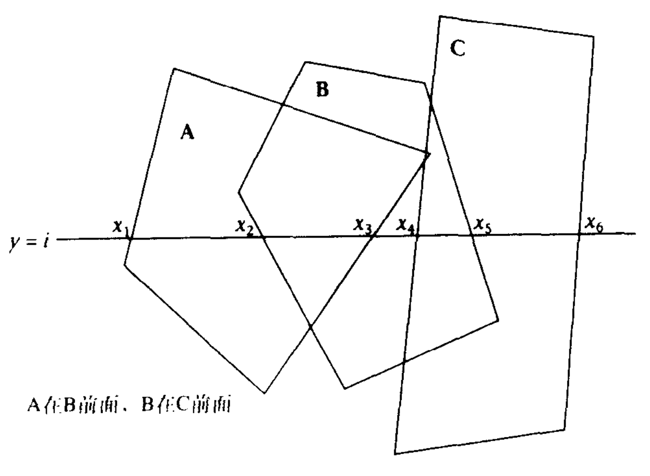


图13-2 多边形扫描线算法

考虑在扫描线$y=i$的情形。这里AET由六个入口所组成，对应于x_1，x_2，…，x_6。从对应于x_1的多边形指针，扫描线从x_1到x_2是属于多边形A的。然而在点x_2处，有两个多边形需要考虑，即A和B。因此x_1和x_2各自多边形指针中的平面方程系数就被用来求解哪个多边形为较近的（通过对每个多边形计算在点（x_2，i）处的z深度值）。此时，多边形A会被认为是最靠近的，

因此直线从x_2到x_3就当作属于多边形A被渲染。从x_3到x_4只属于一个多边形，即多边形B。然而，在x_4处多边形C也必须考虑到，在（x_4，i）点上B和C之间的距离需要类似的确定。继续这样做，每一个扫描线段都得到处理，直到在x_6之后没有其他多边形需要考虑为止。

虽然这个算法在某种意义上看比z缓冲区方式更好，它通常面对很大的多边形数量时速度较慢（比如说超过500个多边形，每个多边形有100个像素的大小）。然而，对于多边形数目不大的情况，它可以比z缓冲区快得多。导致其速度变慢的因素如下：

- 即使是对于凸多边形，当向AET中添加项时，需要排序（因为在一次处理中需要处理所有的边而不单单是一个多边形）；
- 如果多边形的数目很大，在重叠多边形的集合中找出最小z值的多边形，其计算量是相
272 当可观的。

利用扫描线相关性的算法，如名字所暗示的，意味着在扫描线上的水平相关性。然而，一组扫描线之间的相关性也是可以分析的。给定扫描线y的一个有序可见多边形集合，那么对于扫描线 y+1 其可见的多边形有序集合很可能是相同的。这一方面我们可以做更进一步的探讨。一个代价较低（但效率低一点的）方法是对一条扫描线储存多边形的序列，当活动边表处理的是下一条扫描线的时候什么也没有改变，这立刻就可以断定在扫描线上那个多边形片段是可见的，就可以取消对深度计算的需要。实际上，这个方法是受到一定限制的，Crocker（1984）给出了一个更复杂的利用垂直相关性或不可见相关性的算法。

递归细分可见性算法

这是一个分而治之方法，是由 Warnock（1969）提出来的。这种方法是一种在计算机图形学（和其他计算机科学）中经常使用的方法。XY 平面上的裁剪矩形表示了照相机模型的视平面窗口（转换到 PS中）。如果这个矩形窗口所包含的场景其隐藏表面问题很容易被解决的话，那么解决它且渲染场景；否则拆分矩形成四块，对每一块递归地使用相同的原则。

如果确定了场景属于下列四种情况中的哪一种之后，渲染就可以很容易完成：

（1）窗口由一个多边形覆盖，且在场景中没有其他多边形比它更靠近XY平面，那么根据这个多边形所需要的明暗处理渲染该窗口。

（2）没有多边形与窗口相交——那么对窗口无需采取进一步的行动。

（3）只有一个多边形与窗口相交，此时渲染多边形（将窗口作为一个裁剪区域）。

（4）窗口包含一个多边形，此时渲染多边形。

因此算法最初是把整个场景用所需的背景颜色着色，然后对当前窗口进行四个测试。如果所有的测试失败，那么拆分窗口为四个子窗口，然后对这四个窗口的每一个依次递归应用这个过程。

13.3　平滑的明暗处理

Gouraud明暗处理

通过前面的式（6-1），我们给出了基于朗伯定律的明暗处理：

273
$$I = k_a I_a + k_d \sum_{i=1}^{N} I_{pi} \cdot (n \cdot l_i) \tag{13-4}$$

这是针对多个光源的情况。在每个像素点上计算这个表达式代价是很高的，最好避免。另一种方法是双线性插值法，基于式（13-4），只需要计算多边形顶点处的颜色值，然后对z值进行插值。基本的数据结构扩展为：

$$(y2,\ x1,\ Dx,\ z1,\ Dz,\ r1,\ Dr,\ g1,\ Dg,\ b1,\ Db)$$

$$Dr = \frac{r2 - r1}{y2 - y1}$$
$$Dg = \frac{g2 - g1}{y2 - y1} \qquad (13\text{-}5)$$
$$Db = \frac{b2 - b1}{y2 - y1}$$

它表示了一条从（x1，y1，z1，r1，g1，b1）到（x2，y2，z2，r2，g2，b2）的边。对函数Update和函数JoinLines的改变是在相同的线上，如z缓冲区算法。

这个方法称为Gouraud明暗处理，以Gourand（1971）命名，它被用在“平滑的明暗处理”上。假设表面是由一组多边形表示的，当表面有解析表示时我们就可能计算顶点上的正确的多边形表面法向。因此在顶点处的颜色可以使用这些真实的法向来计算，并插值。虽然我们是以多边形的形式进行渲染，但是颜色的插值形成了平滑的明暗表面的效果。

如果表面法向不容易计算出来，那么可以采用一种替代方法，计算在每个顶点处的近似法向，这是通过对顶点所属的每个表面的法向量求平均值得到的。

因为属于一个比较大的多面体对象的邻接多边形有共享顶点，对多边形做颜色插值时，插值使得多个多边形之间的颜色变化变得平滑了。这样我们就不会觉察出多边形边的存在，多面体看起来好像是一个平滑变化的表面，如彩图13-3所示。值得注意的是，当光源紧靠一个大的多边形中心位置的时候，Gouraud明暗处理方法将无法产生可接受的结果。在这种环境下应该发生的现象是，多边形的中心位置处变得特别亮，其边沿处变得比较黑暗。但是真实情况不是这样，因为在中心位置的颜色是通过对顶点处颜色插值得到的，所以想像情形不会发生。为了让Gouraud明暗处理方法生效，多边形的尺寸相对于它们与光源的距离来说要充分得小。当然，这些结论同样适用于任何插值模式。

Phong明暗处理

它在插值模式中加入了镜面反射，而不是像Gouraud明暗处理方法那样只用漫反射。所使用的方程表示如下，这就是最初的式（6-25）：

$$I = k_a I_a + \sum_{i=1}^{N} I_{pi} \cdot ((n \cdot l_i) k_d + (h_i \cdot n)^m k_s) \qquad (13\text{-}6)$$ 274

不像双线性颜色插值，这个方法使用法向的插值。对顶点处的“真”法向插值，然后在每个像素上使用式（13-6）确定颜色。这个方法是由Bui-Tong，Phong（1975）提出来的，参见彩图13-4。

很重要的一点是，对于每一次插值，法向都必须重新规范化（因为给定两个单位法向，它们的插值通常不是一个单位法向）。计算代价是很大的，包括求一个平方根。对于Phong明暗处理有很多加速算法，比如Bishop和Weimer（1986）所提出的方法和Claussen（1989）所提出的方法。Bishop和Weimer对Phong表达式做泰勒展开，来获得一个可用的简化近似值。Claussen注意到如果采用球面坐标形式来表示法向，会得到一个比较好的结果，所以可以对角度进行插值以避免规范化。

13.4 纹理生成

介绍

彩图1-11给出了房间内的一个场景，其中有多个虚拟人物围坐在一张桌子边。我们如何去创造和渲染这种场景呢？场景将创造成一个层次场景图，如我们在第8章中所讨论的。然而，在这样的场景中有很多地方的几何建模是非常困难——尤其是人物的面部、头发和衣服，以及桌椅的表面图案。如果你注意一下窗子，你还会看到外面的景致。这些也必须进行详细建模吗？如果是的话，那么在房间所对应场景部分后面还有其他更复杂的模型存在，其渲染代价将会十分巨大。如何在计算机图形中处理这些精细细节呢？

当然，面部、头发和衣服是可以通过几何方法建模的，但是这会增加巨大数目的多边形到场景图中，由此将在相当大程度上减慢渲染的速度。在一些应用中（特别是那些头发和衣服在其中占有重要地位的应用）这样做可能是值得的。但是假设不是这样，就像我们现在这个例子，应用的目的只是要让场景给人一个好的印象，而且可以肯定的是，虚拟环境的参与者是不可能去近处检查这些精细细节的话，那又该怎么办呢？

这些问题的答案是采用一种称之为*纹理映射*的渲染方法（Catmull，1974；Blinn and Newell，1976）。纹理映射只是一条将某个颜色分配到某个像素的规则——与任何光照模型无
275 关。举例来说，假设在对多边形做扫描线光栅化的过程中，连续的n个像素所构成的块被设定成红色，而下个块被设定为白色，如此这般地渲染多边形各处。然后多边形会是红白色相间的斑纹，斑纹总是沿着相同的方向，与多边形无关。不管怎样斑纹必须附着在多边形上，表示在对象坐标系中，然后斑纹经过渲染管道产生效果——这样斑纹会总是正确地与多边形保持方向。如果多边形被管道裁剪，那么斑纹也会以某种方式被裁剪。最后斑纹颜色与从光照模型所计算出的颜色相结合——是纹理还是光照并不是一个问题，两者可能同时使用。这个简单的例子已经引入了许多纹理映射的思想和问题，我们现在就来详细地加以分析。

上述的例子是一个非常简单的*过程纹理*，也就是说，颜色是由某个公式所确定的（Blinn，1978）。在绝大多数纹理映射应用中“纹理”是一个两维图像，是个颜色值的数组，用来设定像素、储存在计算机内存中。纹理映射的思想是以某种方式在图形对象上“画”这个纹理，以便让它看起来似乎在渲染的时候粘贴在多边形上。换句话说，不再仅仅依照光照模型确定对象所对应的每个像素的颜色，如先前小节所讨论的那样，而是让每个像素的颜色也可以或者干脆由纹理的颜色来确定。因此就要有一些规则来确定所渲染的对象上的每个像素与来自纹理的颜色值之间的对应关系。如何能够完成这一点呢？

纹理像素

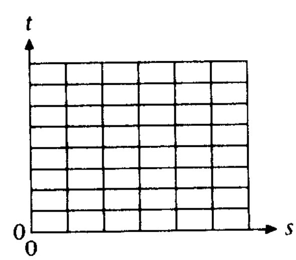


图13-5　纹理像素坐标系

在二维纹理阵列中的每个单个元素称为*纹理像素*。纹理像素有一个颜色，还有一个在行列中的位置。纹理是以坐标（s，t）为参数的，这里 $0 < s, t < 1$，s和t分别表示水平和垂直坐标轴。假设纹理由一个 $M \times N$图像所构成，那么st坐标系统就加在了它上面，
276 在整数表示的纹理像素坐标和st坐标之间存在着一个对应。图13-5给出了一个图像不是正方形这样一种情形。给定任意s和t，我们可以通过下式来求出对应的纹理像素：

$$i = \text{round}(s \times N)$$
$$j = \text{round}(t \times M) \quad (13\text{-}7)$$

这里（i、j）代表一个在第i列第j行上的纹理像素。

映射

让我们考虑一种最简单的情况，多边形为正方形，纹理也为一正方形的映射。两个例子如图13-6所示。在每种情况下，多边形顶点映射到st空间。那么多边形的每个像素就与它覆盖的对应纹理像素相关联，用这个颜色进行渲染。

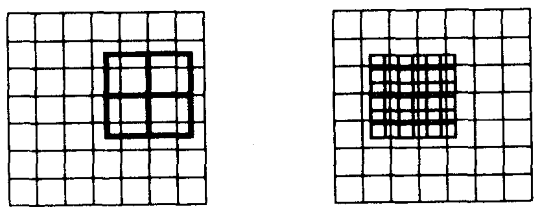


图13-6　像素和纹理像素之间的关系

然而，图13-6说明了情况比这要复杂得多——很少会发生像素分辨率和纹理像素分辨率之间精确匹配的情况。每个像素可能覆盖若干个纹理像素，如在图13-6a中的情况；或者每个像素可能是纹理像素的一小部分，如图13-6b中所示（或两者兼而有之），所以由纹理像素颜色到像素颜色的赋值没有什么简单的规则可循。

在图13-6a中每个像素覆盖多个纹理像素，这称为缩小。此时的问题是应该用哪一个纹理像素的颜色对像素做设定。一条规则是选择中心最靠近像素中心的纹理像素，并设定像素的颜色为那个纹理像素的颜色。它的好处是速度快。缺点是有可能导致在最后渲染的图像上非常严重的走样。另一个方法是设定像素颜色为最靠近它的四个纹理像素的加权平均，权值为它们所覆盖区域的比例。这会变得比较慢，但是会极大地改善图像的效果。放大是另一种情况，它发生在每个纹理像素覆盖多个像素的时候。这里最简单的策略是选择其中心最靠近像素的那个纹理像素的颜色。同样也会出现像素与多个纹理像素重叠的情况，所以加权平均方法在减少走样方面是有用的。

我们知道纹理所要应用到的图形对象是表示在最初对象坐标中的。现在对象被送进渲染管道并最终映射到一组显示像素上（如果没有被完全裁剪掉的话）。假设对象远离投影中心。那么它所占的像素量是很少的，每个像素所对应的对象区域的比例相对较大。因此每个像素相对于纹理像素来说也是大的，覆盖了多个纹理像素。这是缩小现象发生的原因。放大是一种相反的情形，对象位于COP的附近，因此每个像素所占的对象区域是一个非常小的部分，因此它只相当于纹理像素尺寸的一部分大小。 277

注意在上述分析中实际上隐含了两个映射阶段。对象坐标到像素的映射，这与先前一样。同时，对象坐标还要映射到st纹理空间。st纹理空间坐标要进一步映射到纹理像素，通过它们来确定颜色。这需要对象的某种表示以便可以计算到st的映射。首先我们说明这样一种映射对于三角形的情形是可能的。假设三角形有三个非共线的顶点p_0，p_1和p_2。我们可以构造顶点的

一个重心组合来表示三角形。当α_0和α_1扫过它们允许的范围，点p扫过整个三角形：

$$p = \alpha_0 p_0 + \alpha_1 p_1 + (1 - \alpha_0 - \alpha_1) p_2 \tag{13-8}$$

这里，

$0 \leqslant \alpha_0,\ \alpha_1 \leqslant 1$

$0 \leqslant \alpha_0 + \alpha_1 \leqslant 1$

如果我们把α_0当作s，把α_1当作t，那么三角形的每个点p对应到纹理图中的一个位置，如图13-7所示。纹理空间点s=0，t=0 对应于对象空间点p_2，s=1，t=0 对应于p_0，等等。每个属于三角形的点（当然包括它边上的点）对应于纹理空间中惟一点。而且，给定在三角形中的任意点，很容易计算s和t，由此实现这个映射。实际上，该方法需要当三角形光栅化为像素的时候，每个像素坐标一定要转换回对象空间，然后计算相应的纹理坐标，最后得到图像像素所对应的纹理坐标，从而根据所覆盖的纹理像素集合计算像素的颜色。然而，每个像素本身是个有限尺寸的矩形形状。因此在理想情形，像素的角点变换回对象空间，在对象空间形成
278 一个四边形，由此在纹理空间中形成四边形，覆盖一组纹理像素。那么我们就必须综合多个纹理像素的颜色以确定像素的颜色。

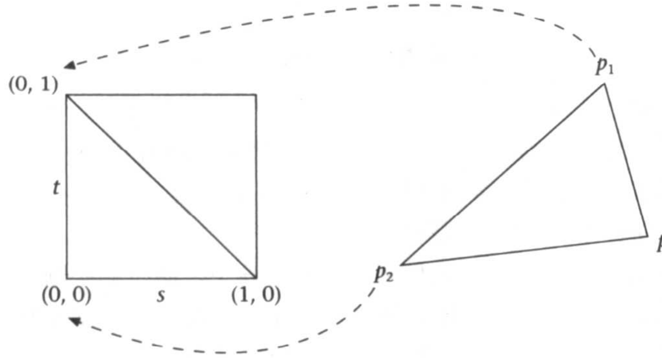


图13-7　从三角形到纹理空间的映射示例

我们使用了这样一个相对简单的例子来说明如何对简单对象（如三角形）做纹理映射。然而，这个特殊例子带有很强的限制性，因为它规定纹理图像要映射到对象空间的部分为三角形，其精确边界可以表示为st空间中的三角形，三个顶点分别为（0，0），（1，0）和（0，1）。



图13-8　纹理空间中的三角形

图13-8给出了一个反例。这里我们希望只将嘴部区域映射到对象空间中一个特定三角形。通常，需要将纹理图像的特定区域映射到特别对象上，而这样的区域通常是任意形状的。实际上，我们需要定义对应于对象空间多边形的每个顶点的纹理坐标，因而只让由这些纹理坐标包围的区域映射到多边形。

对于这种一般情形，用户需要对纹理空间顶点赋值纹理坐标，这些顶点定义了将要映射到对象的部分图像的边界。我们还将以对象空间三角形为例，设三角形的三个三维顶点为 p_0，p_1和p_2。假设纹理坐标（s_0，t_0），（s_1，t_1）和（s_2，t_2）分别对应这些顶点，那么我们需要如下的映射：

$$p_i \to (s_i, t_i), i = 0, 1, 2 \quad (13\text{-}9)$$

还需要一个求对应于三角形中任意点的 *st* 坐标的方法。通常，

$$p = f(s, t), 0 \leqslant s, t \leqslant 1$$
$$p_i = f(s_i, t_i) \quad (13\text{-}10)$$ 279

提供了图形对象的一个参数化形式。我们将在第19章考虑曲线和曲面表示法的时候再具体详细地探讨这种类型关系。对于纹理映射我们还需要逆关系：

$$\begin{aligned} s &= s(x, y, z) = s(p) \\ t &= t(x, y, z) = t(p) \\ (s_i, t_i) &= (s(p_i), t(p_i)) \end{aligned} \quad (13\text{-}11)$$

这里 $p=(x, y, z)$包括对象整个表面（在对象空间中）。通常这些函数是不知道的。然而，通过简化假设，即(s, t)在局部范围内随着p做仿射变化，这样我们就能构造如此映射了。举例来说，假设对象表面是由三角形面片表示的，那么我们能对每个三角形求出映射。此时Heckbert（1986）建议使用下式：

$$x_i = A_x + B_x s_i + C_x t_i, i = 0, 1, 2 \quad (13\text{-}12)$$

这是有三个未知变量的方程组，通过求解方程可以得到未知常数的值。对y_i和z_i可以执行相似的计算，形成如下的变换形式：

$$(x, y, z) = (s, t, 1)\begin{bmatrix} B_x & B_y & B_z \\ C_x & C_y & C_z \\ A_x & A_y & A_z \end{bmatrix} \quad (13\text{-}13)$$

通过对矩阵求逆，这可以被重新表示为式（13-11）的形式。

Watt和Watt（1992）给出了另一个方法（本质上是一样的，但是表示为几何变量形式）。只考虑参数s（对于参数t有类似结论），假设关系有仿射形式如下：

$$\begin{aligned} s_i &= s_0 + (p_i - p_0) \cdot v \\ i &= 1, 2 \end{aligned} \quad (13\text{-}14)$$

这里v是一个向量，与三角形位于同一个平面上，沿着v的方向参数s以最大的速率变化。重写式（13-14）为：

$$s_i - s_0 = (p_i - p_0) \cdot v \quad (13\text{-}15)$$ 280

因此$s_i - s_0$与两矢量（$p_i - p_0$）和v之间的余弦成正比。

图13-9中给出了一些例子。在图13-9a中，s的所有变化是沿着s_0到s_1，因此s的梯度矢量正好对应于$p_1 - p_0$（这个矢量和v的余弦已经取最大值，由此它们之间的角度一定是0）。在图13-9b中，从s_0到s_1的距离与s_0到s_2之间的距离相同，所以v一定在p_0处等分该角度。对于图13-9c有相似的解释。该图从s_0到s_1的距离大于从s_0到s_2的距离，所以v和$p_1 - p_0$之间的角度小于v和$p_2 - p_0$之间的角度。

现在假设 $v=(a_s, b_s, c_s)$，多边形的法向为 $n=(n_x, n_y, n_z)$。那么因为$v \cdot n=0$，根据式（13-14）我们分别有关于三个未知变量a_s，b_s和c_s的三个方程。对变量t有相似的结果，将两者结合在一起得到：

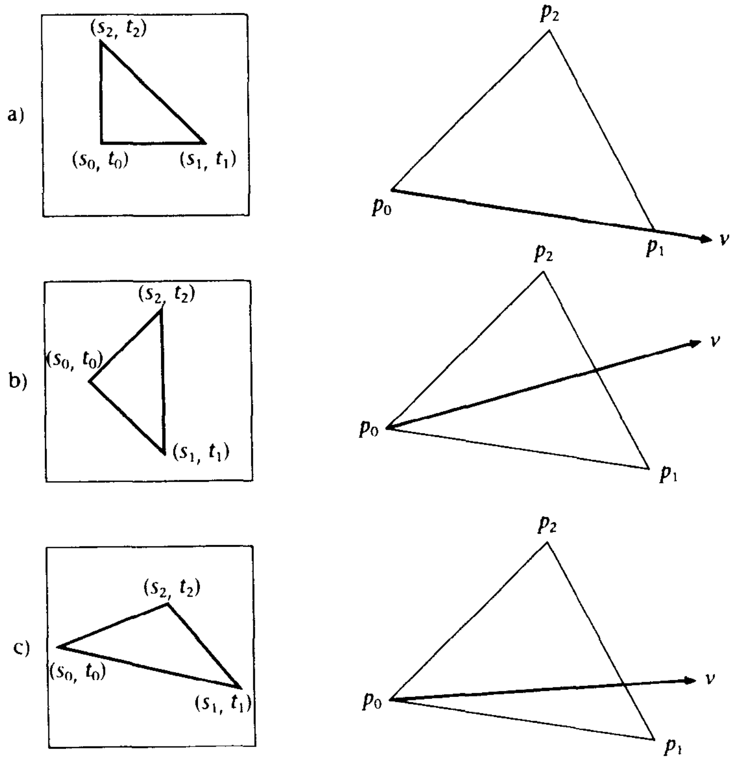


图13-9 s对于纹理三角形不同配置的梯度

$$(s,t)=(s_0,t_0)+(p-p_0)\begin{bmatrix} a_s & a_t \\ b_s & b_t \\ c_s & c_t \end{bmatrix} \tag{13-16}$$

这与式（13-13）具有相同的一般形式。

现在为了概述我们所讲过的纹理映射过程，大的表面要细分成在对象空间中的许多三角形。每个三角形有相关联的纹理坐标以及从对象空间到纹理空间的映射。三角形被送进渲染管道。每个像素的角点被映射回对象空间，到纹理坐标的映射确定了在纹理空间中的四边形。我们计算出覆盖像素点的纹理像素的一个适当的加权平均值，并用在对像素颜色的设定上。

281

仍然存在裁剪问题。假设一个三角形被视景体裁剪并从它的两条边拆分开。结果将会得到一个四边形，如果三角形是惟一可用的图元，这能容易地分割成两个三角形。一个更困难的问题还是在纹理坐标的裁剪。当然被裁剪掉的顶点的纹理坐标不会再被使用。如图13-10所示，顶点p_1位于可见区域的外面。对此有两个可能的解。

每个新的相交点（这里是q_0和q_1）可以被映射回对象空间，然后映射式（13-11）用来求出新的纹理坐标。另一种方法是使用插值。裁剪在透视投影之前进行，因此对实际纹理坐标的裁剪不会造成纹理失真（见下一小节）。现在假设边p_0到p_1的参数化方程为$p(h)=(1-h)p_0+hp_1$，且$p(h_0)=q_0$，那么纹理坐标可以沿着边以相同的比率做插值：

$$\begin{aligned} s_0' &= (1-h_0)s_0 + h_0 s_1 \\ t_0' &= (1-h_0)t_0 + h_0 t_1 \end{aligned} \tag{13-17}$$

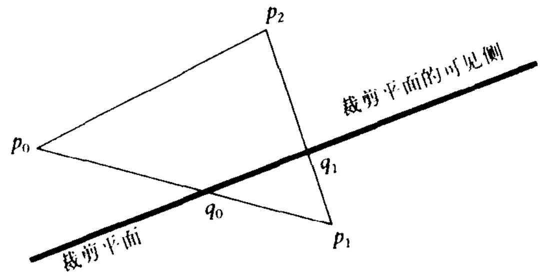


图13-10 经裁剪的三角形的纹理坐标

这里(s_0', t_0')是 q_0的插值纹理坐标。

渐增纹理映射

对每个像素从像素坐标到对象坐标这样的逆变换其代价是相当大的。为此，我们可以审视一下使用如在z缓冲区和插值平滑明暗处理中相同方法的可能性，也就是作为多边形光栅化过程的一部分，渐增地更新纹理坐标。在这个方法中，每个对象空间多边形顶点p_i有它相关联的纹理坐标（s_i、t_i），而且这些是通过渲染管道完成的，它的插值处理如同在 Gouraud 明暗处理中对颜色所采用的方式。然而，通常情况下，如其原来形式，这会产生不正确的结果。为了弄明白这一点，我们集中注意力于对象坐标中线段的简化情况，带有相关联的一维纹理。把这个线段当作多边形的一条边是有帮助的。 282

假设直线段为从p_0到p_1，其参数化形式为：

$$p(s)=p_0+s(p_1-p_0),0\leqslant s\leqslant 1 \tag{13-18}$$

那么s是纹理坐标，不失一般性，我们假设在顶点的纹理坐标是s=0和s=1。

同样不失一般性，我们假设所使用的模型为图9-3中的透视观察模型。投影中心在(0，0，−1)。直线段变成纹理线段，相应的纹理像素与线段空间位置重叠，如图13-11所示。一般来讲，对于这个观察模型，如果(x、y，z)是对象空间中的一个点，那么在投影空间中的对应点如式（13-19）所示。

$$X=\frac{x}{z+1}$$
$$Y=\frac{y}{z+1} \tag{13-19}$$
$$Z=\frac{z}{z+1}$$

那么屏幕上的等价点当然是(X、Y)。因此投影直线段的范围就是从Y_0到Y_1。从图中我们可以很清楚地看到，最初对象空间直线上s表示的同一位置上的区间没有映射到图像平面投影直线上的相同位置区间。因此如果我们通过在纹理坐标0和1之间插值，只对直线上每一个像素简单地选择纹理坐标，将不会得到正确的映射。我们在这个过程中不考虑投影的效果。

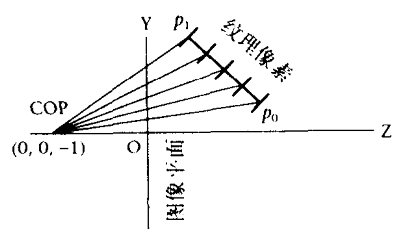


图13-11 将纹理边投影到图像平面

我们要更详细地分析一下这一点。如果这是多边形的一条边，那么它可以表示为一个元组：

$$\left(Y_1, X_0, \frac{\mathrm{d}X}{\mathrm{d}Y}, Z_0, \frac{\mathrm{d}Z}{\mathrm{d}Y}, \cdots, s_0, \frac{\mathrm{d}s}{\mathrm{d}Y}\right) \tag{13-20}$$

因此对应于从Y_0出发的第i条扫描线的屏幕坐标是：

283 $$\left(X_0 + i\cdot\frac{\mathrm{d}X}{\mathrm{d}Y}, Y_0 + i, Z_0 + i\cdot\frac{\mathrm{d}Z}{\mathrm{d}Y}\right) \tag{13-21}$$

根据式（13-19），从屏幕坐标到对象坐标的逆映射为：

$$\begin{aligned} x &= \frac{X}{1-Z} \\ y &= \frac{Y}{1-Z} \\ z &= \frac{Z}{1-Z} \end{aligned} \tag{13-22}$$

把它应用到式（13-21），对应于插值屏幕空间点的对象空间点是：

$$\left(\frac{X_0 + i\cdot\frac{\mathrm{d}X}{\mathrm{d}Y}}{1-\left(Z_0 + i\cdot\frac{\mathrm{d}Z}{\mathrm{d}Y}\right)}, \frac{Y_0 + i}{1-\left(Z_0 + i\cdot\frac{\mathrm{d}Z}{\mathrm{d}Y}\right)}, \frac{Z_0 + i\cdot\frac{\mathrm{d}Z}{\mathrm{d}Y}}{1-\left(Z_0 + i\cdot\frac{\mathrm{d}Z}{\mathrm{d}Y}\right)}\right) \tag{13-23}$$

我们又一次利用式（13-19），依据最初的对象空间坐标重新表达x坐标如下：

$$x' = \frac{(y_1 z_0 - y_0 z_1 + \mathrm{d}y)x_0 + i(z_0+1)(x_1 z_0 - x_0 z_1 + \mathrm{d}x)}{(y_1 z_0 - y_0 z_1 + \mathrm{d}y) - i(z_0+1)\mathrm{d}z} \tag{13-24}$$

对于y坐标和z坐标有相似的表达式。因为直线段的投影是一个直线段，而且投影是可逆的，x'一定是对象空间直线段上的x坐标。因此使用式（13-18），我们可以求出对应的参数值为：

$$s = \frac{x' - x_0}{\mathrm{d}x} \tag{13-25}$$

使用式（13-24）并化简，我们得到：

$$s = \frac{i(z_0+1)^2}{y_1(z_0+1) - y_0(z_1+1) - i\,\mathrm{d}z(z_0+1)} \tag{13-26}$$

这个值是**真纹理坐标**，对应于直线段上的投影点。该直线段是Y_0上方i条扫描线，Y_0为y_0的投影。

现在假设纹理坐标是自插值的，作为直线段渲染的一部分。那么，对于这条边的第i条扫描线，插值纹理坐标应为：

284 $$\begin{aligned} s' &= \frac{i}{\mathrm{d}Y} \\ &= \frac{i}{\frac{y_1}{z_1+1} - \frac{y_0}{z_0+1}} \\ &= \frac{i(z_0+1)(z_1+1)}{y_1(z_0+1) - y_0(z_1+1)} \end{aligned} \tag{13-27}$$

将式（13-27）与式（13-26）比较，我们可以看到直接的纹理坐标插值不能产生正确的结果。插值后的坐标s'和真实坐标 s 之间的关系可以从下式中看出：

$$\frac{1}{s'}=\frac{1}{i}\left(\frac{y_1}{z_1+1}-\frac{y_0}{z_0+1}\right) \tag{13-28}$$

同时有：

$$\frac{1}{s}=\frac{1}{i}\left(\frac{y_1}{z_0+1}-\frac{y_0(z_1+1)}{(z_0+1)^2}\right)-\frac{\mathrm{d}z}{z_0+1} \tag{13-29}$$

因此：

$$\begin{aligned}\left(\frac{z_0+1}{z_1+1}\right)\frac{1}{s}&=\frac{1}{i}\left(\frac{y_1}{z_1+1}-\frac{y_0}{z_0+1}\right)-\frac{\mathrm{d}z}{z_1+1}\\&=\frac{1}{s'}-\frac{\mathrm{d}z}{z_1+1}\end{aligned} \tag{13-30}$$

最后：

$$\frac{z_0+1}{s}=\frac{z_1+1}{s'}-\mathrm{d}z \tag{13-31}$$

从式（13-31）中我们可以看出 $s'\to s$如 $\mathrm{d}z\to 0$。直线段越平，它越是接近于与图像平面平行，插值纹理坐标也就越接近真实纹理。虽然我们是针对单一直线得到的结果，显然这种情形对于多边形每一条扫描线都可以重新生成。因此结果普遍适用于多边形。纹理值的插值对小多边形会产生合理结果，此时深度的变化很小。我们还可以对纹理值插值，然后用一个校正公式如式（13-31）来获得真实的纹理坐标。

为了让插值模式对每一个插值都能产生正确结果，我们执行插值时需要使用齐次坐标。我们还以直线段为例，将纹理坐标表示为（sw，w）的形式。换句话说，为了恢复纹理坐标，用第二个元素除以第一个元素，根据齐次坐标通常的解释。将式（13-26）表示为下列形式：

$$(sw,w)=(i(z_0+1)^2,y_1(z_0+1)-y_0(z_1+1)-i\,\mathrm{d}z(z_0+1)) \tag{13-32}$$

重新表示为屏幕坐标：

$$(sw,w)=\left(\frac{i}{(1-Z_0)^2},\frac{\mathrm{d}Y}{(1-Z_0)(1-Z_1)}-i\left(\frac{\mathrm{d}Z}{(1-Z_0)^2(1-Z_1)}\right)\right) \tag{13-33}$$ 285

现在我们可以用（$1-Z_0$）遍乘，因为这样做不会改变齐次坐标表示的值，有：

$$(sw,w)=\left(\frac{i}{1-Z_0},\frac{\mathrm{d}Y}{1-Z_1}-i\left(\frac{\mathrm{d}Z}{(1-Z_0)(1-Z_1)}\right)\right) \tag{13-34}$$

因此，迭代模式如下所示：

$$\text{Initialize: } sw = 0;\ w = \frac{dY}{1 - Z_1}$$

$$\text{for } i = 0 \ldots dY:$$

$$sw = sw + \frac{1}{1 - Z_0};$$

$$w = w - \left(\frac{dZ}{(1 - Z_0)(1 - Z_1)}\right);$$

$$s = \frac{sw}{w};$$

迭代s开始为 0，结束为 1，正如所要求的。这将留给读者作为练习。

过滤和 mipmapping

我们已经提及了有关缩小和放大的问题——通常这是个对纹理像素过滤的问题，由此来产生对应像素的合理颜色。这对于避免严重走样现象发生十分关键。这样的走样对于单一图像可能是可接受的，但是对动画就是完全无法接受了——因为对象和视点是到处移动的，纹理会闪烁而且扭曲，而不是在对象的表面上无变化。

有很多种过滤的方法，Heckbert（1986）对此有详细的讨论。在OpenGL中最简单的方法是选择坐标最接近于渲染像素中心的纹理像素坐标。这是计算代价最小的方法，但它也是一种最容易产生严重走样的方法。另外一种方法是找到最靠近像素中心的四个纹理像素（一个2×2 数组），对这些颜色做线性插值。

OpenGL也提供了一种被称为*mipmapping*的方法（Blinn and Newell，1976），这是克服缩小和放大所引起的问题的一个高效方法。其思想是储存纹理图像的多个版本，每个版本具有不同的分辨率。假设最高分辨率纹理映射（第0层）的大小为$2^m \times 2^n$。对这个纹理图像每个 2 × 2 的纹理像素块做平均，得到一个新的纹理图像（第1层），它的分辨率为 $2^{m-1} \times 2^{n-1}$。将这样相同的平均过程重复下去直到最后产生一个1 × 1的 纹理图像。如果 $m>n$，这将是第m层纹理。基于对象在屏幕上的尺寸，我们可以在渲染过程期间动态地选择适当分辨率的纹理。过滤和 mipmapping 可以被结合在一起使用。在放大情形中，像素尺寸是纹理像素尺寸的一个部分，此时只有使用最高分辨率的纹理（ 第0层 ）。

对于缩小情形，插值可以在各个 mipmap图像上进行，同时也要在每个mipmap 里进行，
286 避免在mipmap图像间纹理分辨率的改变所带来的变动。OpenGL 提供了如下一些可能性（Neider et al.，1993）：

- 在任何 mipmap的内部，选择最近的纹理像素，或者对最近的2 × 2 方块做线性插值，如前。
- 对两个“最近的”mipmap中的每一个选出最靠近的纹理像素的中心，对它们做线性插值。
- 对两个 “最近的”mipmap中的每一个，插值2 × 2 最近的方块，并在mipmap间对所产生的结果值再插值。

上面是假设从可用的mipmap图像离散集合到对象的连续范围尺寸（即缩小度）之间基本上是存在线性映射的。因此“最近”这个词经常使用在映射的上下文中。

选择纹理坐标的方法

在所有的上述讨论中我们都是针对单一对象进行的——实际上是一个多边形，或更明确

地讲是一个三角形。分配纹理坐标到单一多边形上相对来讲是容易的，如同我们所看到的那样。然而，这不是很有用的。我们需要对彩图1-11中所示的桌子表面铺纹理，而不是对其中的每个面铺纹理。每个面是一个多边形网格，而一个纹理需要应用到整个网格上。纹理坐标选择方式要保证对整个网格有一个全面和一致的纹理构造。

对这个问题没有一个“准确的”回答。一些类型的表面有自然的参数化形式（如Bézier曲面和B样条曲面，这些将在第19章中讨论）。但是多边形网格没有这样一种参数化形式，我们不得不以某种方式选择一个从顶点到纹理坐标的映射。Watt和Watt（1992）对所使用的各种方法作了一个很好的回顾。基本思想是选择一个确有自然参数化形式的表面，而且它计算效率高且容易使用。然后让这个表面包裹住多边形网格，采用投影方法将网格顶点投影到包围表面上。包围表面的纹理坐标然后被用于顶点的纹理坐标。

我们用一种常用技术即基于圆柱体的方法来说明这一点。不失一般性，假设圆柱体的半径为r、中心位于原点上、高度为h，这些将用于包围表面。这样，圆柱体的方程为: 287

$$
\begin{aligned}
x(\theta)&=r\cos\theta\\
y(\theta)&=r\sin\theta\\
0&\leqslant\theta\leqslant 2\pi\\
0&\leqslant z\leqslant h
\end{aligned}
\tag{13-35}
$$

圆柱体可以扁平化成一个矩形，宽度为$2\pi r$，高度为h。因此自然参数化形式是:

$$
\begin{aligned}
s&=\frac{\theta}{2\pi}\\
t&=\frac{z}{h}
\end{aligned}
\tag{13-36}
$$

其中$0\leqslant s$、$t\leqslant 1$。现在给定一个凸多边形网格，在它的顶点和包围圆柱体之间可以定义一个映射。这可以用图13-12中的二维形式说明。这里顶点被投影到圆柱体，投影保证从顶点出发的光线与圆柱体表面垂直。相交点可以用来计算（s，t）纹理坐标。

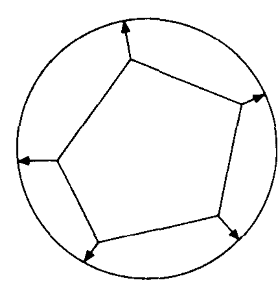


图13-12 从网格顶点到圆柱体纹理坐标的映射

用圆柱体表示多面体网格是简单的，虽然对于在“顶面”和“底面”上的多边形极端情形存在问题——那儿没有圆柱体部分可以匹配。另一种方法是使用包围球体。同样可以从式（2-20）构造出简单表示。我们可以使用放射状投影，即把球心当作投影中心，将顶点投影到球体表面。这里的问题是映射不能在球面上产生一致的参数化形式，所以纹理在它的极上将会发生“压扁”现象。

建立从多边形网格顶点到纹理坐标的映射多少包含一点艺术的味道——通常这是需要不断实验和修正的。

OpenGL实例

在这一小节中让我们来看一个简单的OpenGL程序，它为第8章中的一个例子添加纹理， 288
该例子是构造一堆立方体，一个位于另一个的上面。一个简单的纹理添加到立方体上。程序中的注解是无需过多解释的。

```
#include <GL/glut.h>
/*
  An example of using OpenGL to render a simple object with
  a texture. The texture is saved in a ppm format file.
  You can also use the program dmconvert to tell you more about
  the image file.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*This next is referred to in several places, and because of the callback
interface, cannot be passed as a parameter, hence, global*/

/*WARNING - using the PPM file format, you must delete the first few
lines of text, starting only from the image size data*/
typedef GLubyte Pixel[3]; /*represents red green blue*/

int Width, Height; /*of image*/

/*array of pixels*/
Pixel *Image;

/*name of image file*/
/*char Filename[30];*/
char *Filename = "../filename.ppm";

int allowedSize(int x)
/*returns max power of 2 <= x*/
{
  int r;

  r = 1;
  while(r < x) r=(r<<1);

  if(r==x) return r;
  else return r>>1;
}

void readImage(void)
/*reads the image file assumes ppm format*/
{

  int w,h,max;
  int i,j;
  unsigned int r,g,b;
  int k;

  FILE *fp;

  fp = fopen(Filename,"r");

  /*read the width*/
  fscanf(fp,"%d",&w);
  /*read the height*/
  fscanf(fp,"%d",&h);

289  /*I think that this is max intensity - not used here*/
  fscanf(fp,"%d",&max);

  /*width and height must be powers of 2 - taking the simple option
  here of finding the max power of 2 <= w and h*/

  Width = allowedSize(w);
```

```
  Height = allowedSize(h);

  printf("filename = %s\n",Filename);
  printf("Width = %d, Height = %d\n",Width,Height);

  Image = (Pixel *)malloc(Width*Height*sizeof(Pixel));

  for(i=0;i<Height;++i){
    for(j=0;j<Width;++j) {
      fscanf(fp,"%d %d %d",&r,&g,&b);
      k = i*Width+j; /*ok, can be more efficient here!*/
      (*(Image+k))[0] = (GLubyte)r;
      (*(Image+k))[1] = (GLubyte)g;
      (*(Image+k))[2] = (GLubyte)b;
    }
    /*better scan to the end of the row*/
    for(j=Width; j<w; ++j) fscanf(fp,"%c %c %c",&r,&g,&b);
  }
  fclose(fp);
}
```

函数readImage通过指针Filename读取一个文件中的图像。图像假设储存为PPM（可移动pixmap 格式），包含宽度、高度、最大光强度，后面紧跟的是每个像素的由三元组表示的RGB颜色值，以字节形式储存。注意在PPM文件的开头还有一些附加的信息——但是为了保持例子的简洁这些已经被删除掉了。图像存储在Pixel数组Image中。

```
void initializeTextures(void)
{
  GLint level = 0;    /*only one level - no level of detail*/
  GLint components = 3; /*3 means R, G, and B components only*/
  GLint border = 0;    /*no border around the image*/

  /*read the image file*/
  readImage();

  /*each pixelrow on a byte alignment boundary*/

  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  /*define information about the image*/
  glTexImage2D(GL_TEXTURE_2D,level,components,
    (GLsizei)Width, (GLsizei)Height,
    border, GL_RGB, GL_UNSIGNED_BYTE,Image);

  /*ensures that image is not wrapped*/
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);

  /*chooses mapping type from texels to pixels*/
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  /*this chooses the texel for minification and magnification
  GL_NEAREST chooses the texel nearest the center of the pixel.
  GL_LINEAR performs a linear interpolation on the 4 surrounding
  texels*/

  /*GL_DECAL - this says overwrite pixel with texture color*/
  glTexEnvf(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_DECAL);
  /*an alternative is GL_MODULATE which modulates the lighting
  by the texel value by multiplication*/
```

290

```
    /*this enables texturing*/
    glEnable(GL_TEXTURE_2D);
  }
```

函数initializeTextures实际读取图像，并使用glTexImage2D声明其中的信息。在这个例子中没有用到mipmap，所以只定义了一层图像（第0层）。图像没有指定边界，像素是以 RGB 无符号字节值进行插值的。当纹理坐标跑出范围 [0, 1]的时候，一种可能性是“回绕”，这样纹理以平铺的方式覆盖在对象表面。在这里我们保证没有回绕，图像好像是堆在对象表面上。我们说明在处理过滤上的选择。程序使用包含四个最近的相邻纹理像素的方块之间的线性插值作为像素的中心。另一种方法（GL_NEAREST）仅仅选择最近的纹理像素。

纹理颜色与像素的结合可以有各种不同的方式。它可能完全取决于像素颜色，通过使用GL_DECAL参数。另一种方法是通过乘法运算用纹理调制光照计算值。最后纹理映射通过函数glEnable (GL_TEXTURE_2D) 激活。

291

```
static void cubebase(void)
/*specifies a side of a cube*/
{
  glBegin(GL_POLYGON);
    glTexCoord2f(0.0,0.0);
    glVertex3d(-0.5,-0.5,0.0);

    glTexCoord2f(0.0,1.0);
    glVertex3d(-0.5,0.5,0.0);
    glTexCoord2f(1.0,1.0);
    glVertex3d(0.5,0.5,0.0);

    glTexCoord2f(1.0,0.0);
    glVertex3d(0.5,-0.5,0.0);
  glEnd();
}
```

函数cubebase定义对象空间立方体的基。然而，注意到纹理坐标是根据顶点给出的。整个纹理空间被映射到基上。

```
static void cube(void)
/*uses cube side to construct a cube, making use of the modelview matrix*/
{
  /*make sure we're dealing with modelview matrix*/
  glMatrixMode(GL_MODELVIEW);

  /*pushes and duplicates current matrix*/
  glPushMatrix();

  /*construct the base*/
  cubebase();

  glPushMatrix();
  /*construct side on +x axis*/
  glTranslated(0.5,0.0,0.5);
  glRotated(90.0,0.0,1.0,0.0);
  cubebase();

  glPopMatrix();

  /*construct side on -x axis*/
  glPushMatrix();
  glTranslated(-0.5,0.0,0.5);
```

```
   glRotated(-90.0,0.0,1.0,0.0);
   cubebase();
   glPopMatrix();

   /*construct side on +y axis*/
   glPushMatrix();
   glTranslated(0.0,0.5,0.5);
   glRotated(-90.0,1.0,0.0,0.0);
   cubebase();
   glPopMatrix();

   /*construct side on -y axis*/
   glPushMatrix();
   glTranslated(0.0,-0.5,0.5);
   glRotated(90.0,1.0,0.0,0.0);
   cubebase();
   glPopMatrix();
   /*construct top*/

   glBegin(GL_POLYGON);
      glTexCoord2f(0.0,0.0);
      glVertex3d(-0.5,-0.5,1.0);

      glTexCoord2f(1.0,0.0);
      glVertex3d(0.5,-0.5,1.0);

      glTexCoord2f(1.0,1.0);
      glVertex3d(0.5,0.5,1.0);

      glTexCoord2f(0.0,1.0);
      glVertex3d(-0.5,0.5,1.0);
   glEnd();

   glPopMatrix();

   glFlush();
}

static void stack(int n)
/*creates a smaller cube on top of larger one*/
{

   cube();
   if(n==0)return;

   glPushMatrix();
   glTranslated(0.0,0.0,1.0);
   glScaled(0.5,0.5,0.5);
   stack(n-1);
   glPopMatrix();
}
```

292

函数cube如我们先前所定义的——通过主基的复制并旋转构造出来。函数stack递归地产生层次结构，如前所述。程序输出的一个例子如彩图13-13。

13.5 VRML97例子

如这一章中所述，对于建模者来说主要困难在于为几何形状的顶点选择纹理坐标。对于每个基本体素，有一个预先定义的纹理坐标集合将纹理指定到每个面。对于IndexedFaceSet

几何节点，必须定义每个顶点的纹理坐标。

对纹理映射选择的完整介绍超出了我们这里的范围。用于纹理映射的实际图像可以定义为单信道、三信道或四信道图像。如果使用一个成分（灰度图像），那么我们就用它来调制多
293 边形的基颜色。如果使用三个或四个成分的图像，那么纹理颜色替换了多边形的基颜色。第四个成分如果存在的话，我们把它作为透明度信道，或α信道。图像源是通过三个纹理节点之一定义的。它可以是一个外部图像文件（ImageTexture）、一个外部电影（MovieTexture）或在VRML文件中直接定义的图像（PixelTexture）。对于ImageTexture节点，一般倾向于使用PNG和JPEG格式文件。

VRML97提供了一个应用二维变换到对象纹理坐标的设施，即使用Texture Transfromation节点。纹理和纹理变换节点都是位于外观节点里面的一个域。图13-14a给出了在动作中纹理变换的一些简单例子。通过纹理变换可以伸展纹理，旋转或移动纹理到方盒表面上的其他位置（见彩图13-14b）。

```
#VRML V2.0 utf8
Transform {
 translation 0 0 0
 children [
  Shape {
  appearance Appearance {
    texture DEF my_tex ImageTexture {
     url ["default.jpg"]
    }
   }
   geometry DEF my_box Box {
    size 2 2 2
   }
  }
 ]
}
Transform {
 translation 3 0 0
  children [
   Shape {
    appearance Appearance {
     texture USE my_tex
     textureTransform TextureTransform {
      scale 3.0 3.0
     }
    }
    geometry USE my_box
   }
  ]
 }
Transform {
 translation -3 0 0
 children [
  Shape {
   appearance Appearance {
    texture USE my_tex
    textureTransform TextureTransform {
     translation 0.5 0.5
    }
   }
   geometry USE my_box
  }
 ]
}

Transform {
 translation 0 3 0
 children [
  Shape {
   appearance Appearance {
    texture USE my_tex
    textureTransform TextureTransform {
     rotation 0.785
    }
   }
   geometry USE my_box
  }
 ]
}
```

294 图13-14　a) 纹理映射的基本体素实例

在IndexedFaceSet节点上应用纹理坐标有两种主要方式。每个顶点可能有多个纹理坐标与之对应。这是因为每个顶点可能属于多个相交于该点的面，在每一个面上该顶点对应于一个纹理坐标，这些纹理坐标可能是相同的，也可能是完全不同的。一般来讲，当用一个纹理来包裹整个对象的时候，一个顶点在每个面中将有相同的纹理坐标。在两种情形中，IndexedFaceSet的texCoord域是需要的。如果顶点在每个面上有相同的纹理坐标，那么纹理坐标可以通过coordIndex域数组中的指针进行索引。这样在texCoord域中的纹理坐标数目一定等于coord域中坐标的数目。

如果顶点在每个面中有不同的纹理坐标，那么在texCoordIndex域中的数组必须设定。这个数组的长度与coordIndex域中的数组长度是一样的。

图13-15a给出了这两个纹理应用方法的例子。两个立方体通过手工从点的序列开始建造。对于第一个立方体，纹理映射是根据每一个顶点每一个面的纹理坐标来定义的。对于第二个立方体，每个顶点有惟一的纹理坐标。对于第二个立方体，我们必须将形状拆分成两个比较小的形状，以便得到一个切合实际的纹理映射（见彩图13-15b）。

```
#VRML V2.0 utf8

Transform {
 translation -1.5 0 0
 children [
  Shape {
   appearance DEF my_app Appearance {
    texture ImageTexture {
     url ["default.jpg"]
    }
   }
   geometry IndexedFaceSet {
    coord DEF my_coord Coordinate {
     point [-1 0 -1, 1 0 -1, 1 0 1, -1 0 1,
            -1 2 -1, 1 2 -1, 1 2 1, -1 2 1]
    }
    texCoord TextureCoordinate  {
     point [0 0, 0.333 0, 0.666 0, 1 0,
            0 1, 0.333 1, 0.666 1, 1 1]
    }
    coordIndex [0 1 2 3 -1, 1 5 6 2 -1,
     3 2 6 7 -1, 0 3 7 4 -1,
     0 4 5 1 -1, 4 7 6 5]
    texCoordIndex [6 7 3 2 -1, 1 5 6 2 -1,
     3 2 6 7 -1, 6 2 1 5 -1,
     0 4 5 1 -1, 5 1 0 4]
   }
  }
 ]
}
Transform {
 translation 1.5 0 0
 children [
  Shape {
   appearance USE my_app
   geometry IndexedFaceSet {
    coord USE my_coord
    texCoord TextureCoordinate  {
     point [0 0, 0.333 0, 0.666 0, 1 0,
            0 1, 0.333 1, 0.666 1, 1 1]
    }
    coordIndex [1 5 6 2 -1, 3 2 6 7 -1,
     0 4 5 1]
   }
  }
  Shape {
  appearance USE my_app
   geometry IndexedFaceSet {
    coord USE my_coord
    texCoord TextureCoordinate  {
     point [0.666 1, 1 1, 1 0, 0.666 0,
            0.333 1, 0 1, 0 0, 0.333 0 ]
    }
    coordIndex [0 1 2 3 -1, 0 3 7 4 -1,
     4 7 6 5]
   }
  }
 ]
}
```

图13-15　a) 纹理映射的IndexedFaceSet 实例

13.6 小结

本章的基本目标是完成渲染管道的构造——包括图像空间可见性、带有平滑明暗处理的局部照明，以及纹理生成。我们看到了多边形填充算法可以以非常有效的方式处理所有这些 295
内容。同时也看到了渲染管道是由两个基本阶段所组成：几何管道和渲染过程。第一个阶段处理在局部坐标系中表示对象几何属性的场景图。对象之间的关系表达为图的结构（拓扑信息）和局部变换矩阵（几何信息）。将多边形转换到投影空间，经过建模和观察变换，然后进行裁剪和投影。渲染过程处理从几何管道输出单个多边形，并通过多边形填充算法渲染它们，且要考虑到z缓冲区、明暗处理和纹理生成信息。多边形的填充显然可以以非常高效的方式实现——尤其最初多边形经过了三角化处理后。甚至不需要保存边表（因为每一条扫描线每个多边形只有两个交点，不存在局部最小和最大），但是扫描线相关性显然是需要维护的。

这样我们就已经完成了从“真实”到“实时”旅程的第一个阶段。我们开始于先前章中的光线跟踪（包括真实感元素），摒弃了那些获得实时性能的光线跟踪内容，现在我们获得了一个比较切实的实时解决方案，但是这样所产生的图像本质上是基于局部照明的。在下一章中我们将说明该如何增加阴影到场景中。在下下一章中我们完成这个循环，介绍一种新的方法——即辐射度方法。它提供（另一种不同类型的）真实感，采用全局照明的方式，而且静 296
态场景可以在实时漫游中渲染。

第三部分　从实时到真实

第14章　阴影：达到实时性真实

14.1　引言

在第5章中讨论的光线投射和第6章中讨论的光线跟踪本身都包含了阴影的渲染。你可能记得我们曾经从光线与对象上任意一个相交点开始跟踪“阴影感知器”到场景中的（点）光源集合。那些到达光源而没有受到来自另外对象阻碍的光线对局部光照有贡献。那些受到其他对象阻碍而没能够到达光源的光线对光照没有什么帮助。显然阴影就是通过这个过程自动生成的。

当我们放松对于全局光照的要求，从光线跟踪转变到渲染管道的时候，我们失去了阴影。然而阴影能够为场景增添极大的真实感程度。本章将考虑是否可以将阴影重新加入到场景渲染中，同时又能保持实时渲染的可能性。

有趣的一点是，我们注意到可见性问题与阴影之间存在着密切的关系。“阴影”的最基本含义是场景中光照射不到的那个部分。如果我们考虑的光是单一点光源（或者是线光源），阴影的思想是正确的，因为每当分析关于观察者的可见性的时候，我们也至少考虑到了部分阴影的问题——用相对于光源的可见性取代相对于观察者的可见性。然而，现实中的光源可不仅仅是点，它们都有一定的区域。虽然在这种情况下我们仍可以把阴影计算问题看成是可见性问题，但是它已经变成了一个更加困难的问题了。此时将要考虑在某个体积区域里从任意点哪些将是可见的。 297

在真实世界中，光源发光表面是一个非零大小的区域（*面光源*）。由这样的光源所产生的阴影强度从被照射到的地点到处于阴影中的地点是逐渐地改变的。它们可以被区分为两个区域。最里面的没有接受任何光的部分称为*本影*，而外部区域称为*半影*。求出本影和半影之间的精确边界以及半影中每一个点的强度是一件非常困难且计算强度很大的事情。我们在稍后部分将会看到在半影中的强度不是连续改变的。

在计算机图形学中我们通常是要去简化阴影确定问题。如果目标是构造一个实时系统，那么我们是不能够负担面光源代价的。实际上，我们总是假设光源是个数学点（*点光源*）或者认为它位于无限远的地方（*线光源*）。由这种光源所产生的阴影没有半影。此时阴影有一个定义明确的边界。本影可以假定它的强度到处都是一样的，因为它是完全不受光源照射的。这种阴影看起来很硬，甚至明显有人造的痕迹；然而，它们仍然能有效地提供空间线索，而且使图像更加有真实感（见图14-1）。

阴影的一个非常重要的属性是视图独立。阴影只依赖于场景中对象和光源的几何属性，当我们改变观察参数时它们并不改变。这一点使得我们可以预先计算静态场景的某些或者是所有的阴影信息。

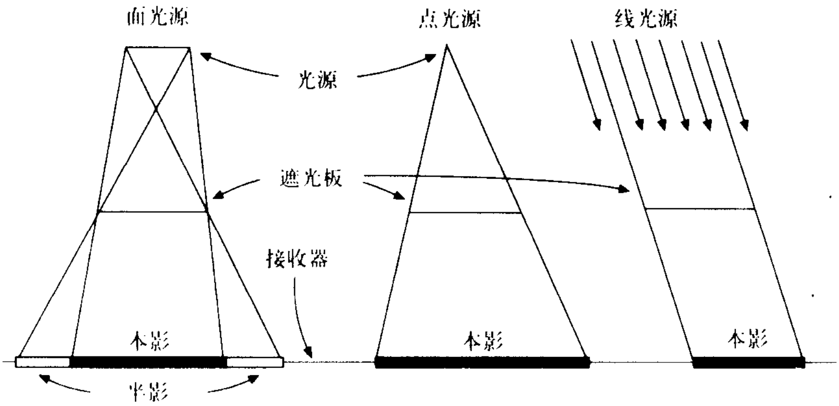


图14-1 柔和的阴影和硬的阴影

在本章的其余部分，我们将使用遮光板这个术语表示引起阴影的多边形，使用接收器这
298 个术语表示阴影所投射到的多边形。

我们将从硬阴影（阴影本影）生成算法的分析开始，这也包括通常实际中所采用的局部解（伪阴影）。然后我们将继续探讨柔和阴影问题。目标是介绍一些有代表性的算法，通过这些算法来阐述阴影生成的一些思想。Woo等（1990）对此有一个比较全面的综述，有兴趣的读者可以做进一步研究。

14.2 阴影本影

一般方法

计算来自点光源的阴影，即使它不是很真实，通常是我们所能做到的最好结果。因为这些方法与可见表面确定方法（VSD）（第11章和第13章）非常相似，我们将依照相同的分类。我们将不同方法——存在大量这种方法——分别分类为图像精确方法、对象精确方法以及混合方法。

图像精确方法在像素层次上计算阴影信息。我们已经看到过这种方法，即光线跟踪。在这一章中我们将要研究这一类中的另外一种方法，也称为阴影缓冲区方法。这一种类也可以包括扫描线方法，这里阴影边被投影到待显示的多边形上，在扫描线过程中这种阴影边是阴影与非阴影之间转换的标志（Appel，1968；Bouknight and Kelley，1970）。即使计算并不是严格地按每一像素进行的，但是它仍然是图像精确级别的。

对象精确方法独立于视点计算阴影信息，通常与几何信息同存储于一个数据库中。既然
299 阴影信息无论从哪个观察角度看都是一样的，所以只要几何属性保持静态不变就无需重新计算。Atherton 等（1978）描述了这种方法中最早的方法之一。它是两遍隐藏表面算法。在第一遍中，光源位置被当作视点，区分多边形中光源所能看见的部分，那些看不见的部分就位于阴影中了。将阴影多边形添加到最初的场景中，在第二遍中场景从照相机视点开始遍历，删除隐藏的表面并执行渲染过程。两遍中它们都使用了基于一般多边形裁剪的VSD算法。Slater（1992a）提出了一个一般性的细分空间方法，以平铺立方体的形式求多边形之间的阴影关系，然后使用阴影体于这些关系上以求出精确阴影。这类似于Haines和Greenberg（1986）所提出的光缓冲区方法，这是用于光线跟踪上下文中的一种方法。这一类中的另外一个算法

是基于二叉空间分割，也称为阴影体 BSP（SVBSP）树方法，是由Chin和 Feiner（1989）提出来的。我们将会在本章稍后部分更详细地研究这个方法。即使阴影是在预处理阶段计算的，也并不意味着在这些方法中的阴影无需帧频的代价，因为我们可能增加了场景的复杂度。

混合方法一部分是在对象精确复杂度中工作，一部分是在图像精确复杂度中工作。我们这里将要看到的例子阴影体（Crow，1977；Bergeron，1986）是最老同时也是最流行的方法之一。在这个方法中，阴影所遮盖的空间体积（阴影体）是在对象空间中计算的，只要场景是静态的就保持有效，但是在表面上的最后阴影确定是根据逐个像素和每个给定视点计算出来的。

最后我们要讨论一下“伪阴影”（Blinn，1988）。这个方法只能计算阴影的一个子集——那些位于一个平面表面上的阴影——同时它也是最容易实现的，而且对于动态对象来说是限制性最小的一种方法，因为它在每一帧中都是完全重新计算的。

阴影z缓冲区

这个方法首先是由Williams（1978）提出来的。它是一个两步方法。在第一步中，我们使用光源作为视点并渲染场景到 z缓冲区之中。注意在这个步骤中对颜色缓冲区的内容不感兴趣，只是对每个像素相对于光源的深度信息感兴趣。可以关掉光照和纹理生成以加速这一过程。我们只保存z缓冲区，称之为*阴影深度缓冲区*或*阴影图*。这个阴影图可以被反复使用，只要我们不改变场景的几何属性和不移动光源。我们称由光源所定义的坐标系统为*光空间*。

在第二步中，我们像往常一样从视点渲染对象，惟一不同的是在扫描转换它们的时候，我们在写每一个可见对象之前要对它做阴影测试。为了做这一点，我们将像素的坐标（x_v，y_v，z_v）从观察空间转换到光空间坐标（x_s，y_s，z_s）（见图14-2），并比较z_s值与存储在阴影深度缓冲区内（x_s，y_s）位置上的深度值。如果像素的深度大于所存储的深度，那就意味着从光源处看，有较靠近的对象遮挡了它，因而它将位于阴影中。如果z_s值等于阴影缓冲区中的值，那说明它就是从光源处所能看到的那个点。 300

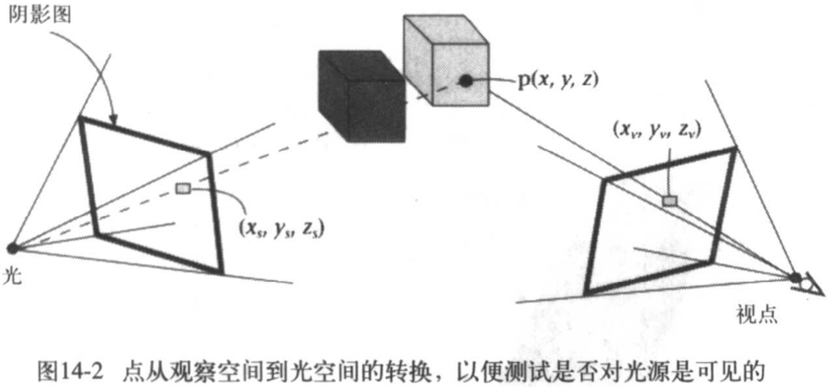


图14-2　点从观察空间到光空间的转换，以便测试是否对光源是可见的

这个方法的好处之一是它不局限于多边形模型。任何可以扫描转换到z缓冲区的表面都可以用于求阴影。另一方面，因为图像精确特性，它容易产生走样和量化误差。误差的一个来源是为深度比较所做的点的变换，也就是从观察空间到光空间的变换。由于有限的精度，z值的比较可能导致在“相等”的时候出现“大于”的情况，这就是大家所知道的“shadow ackne”，即在被照射表面上出现随机的亮点。这个效果可以通过在比较中引入一个公差达到最小化。

过滤和抖动可以用来尽量减少其他的走样问题（Williams，1978；Reeves et al.，1987）。

上面所述的方法执行代价是很高的，因为它必须对要写入颜色缓冲区中的每个像素做阴影比较。许多时间被浪费掉了，因为我们所测试的很多像素将会在可见性 z缓冲区过程期间被重写，不会出现在最后的图像中。我们能做的一种加速方法是延期进行阴影测试直到完成了对图像的渲染。此时阴影计算的第二个步骤是独立于场景复杂度的。然而，我们会在图像质量上付出代价，这是因为我们对图像像素已经执行了光照操作，所有能做的就是对每个值都减去一个常量，这样就能避免在阴影区中出现高光。

如果我们采取的是上面所提到的后一种方法，虽然每一帧阴影计算是与场景复杂度无关的，我们仍然需要执行大量的单个像素比较，而这个代价是相当高的。Segal 等（1992）介绍了这个方法的多个变种方法，它们都是使用纹理映射硬件来进行比较运算的。硬件变换能非常快速地运行，但需要特殊纹理生成硬件，而这并不容易得到。

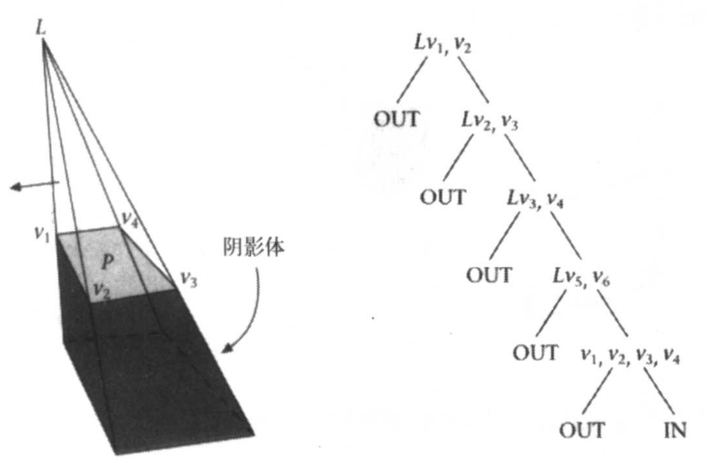
多重光源的模拟可以通过为每个光源提供一个阴影缓冲区来实现。如果我们采取后处理方式，那么可以对每个光源顺序地进行，使得同一个缓冲区用于所有的光源；然而，对每个帧的阴影图像是必须重新计算的。

阴影体

利用阴影体思想的算法有很多。在这一小节中我们将研究一种由Crow(1977)所提出来的混合方法，在下一节中还将看到一种对象精确方法，即SVBSP树。

针对于点光源，多边形阴影体(SV)是多边形后面光线所照射不到的空间体。它是一个由阴影平面(SP)所定义的半无限金字塔，其上表面是多边形本身。给定一个光源L和一个多边形

301 P，多边形P由顶点序列 $[v_1, v_2, \cdots, v_n]$定义，如图14-3a中所示，阴影平面由三元组（L，v_i，v_{i+1}）定义，这里i=1，⋯，n，n+1（$v_{n+1} \equiv v_1$）。阴影体P定义为一个棱台，它的上下表面分别是P和阴影平面(P，L)。



a) 阴影体　　b) 阴影体的BSP表示

图　14-3

当计算位于阴影体内部的平面的时候，很重要的一点是要让多边形平面方程能正确地表示平面的“前侧”和“后侧”。采用右手定则，即正对平面的前侧看去，其顶点描述为反时针

方向的顺序，如图14-3中所示。那么对于图中情形，计算阴影平面方程的正确顶点顺序是v_2Lv_1。此时多边形用顶点描述为v_1，v_2，v_3，v_4，这样反时针顺序使得它是面向光源的。我们所采用的一条约定是，阴影平面的“背面”位于阴影一侧。

阴影体方法首先是由 Crow（1977）提出来的，后来经过 Bergeron（1986）的修改，至今它仍然是阴影计算最流行的方法之一。它基于下列思想，如图14-4 所示。假设有一个不位于阴影中的视点，如果从眼睛处向场景中的某个点p画一个矢量v，我们确定该点是否在阴影中的方法是看v在到达p之前所相交的前向（front-facing）阴影平面数和背向（back-facing）阴影平面数。如果前向阴影平面数和背向阴影平面数之差等于零，那么该点是光线能照射到的（图14-4中的p_2），否则它在阴影中（图14-4中的p_1）。注意，这个差值可能大于1，因为某些区域可能位于多个多边形的阴影体中（图14-4中的p_3）。眼睛位于阴影中是一个特殊的情形。我们需要修改计数规则以便考虑到这一情形。

算法分两个步骤进行：

（1）在对象空间中构造阴影体。 302

（2）通过阴影平面计数确定每个像素阴影。

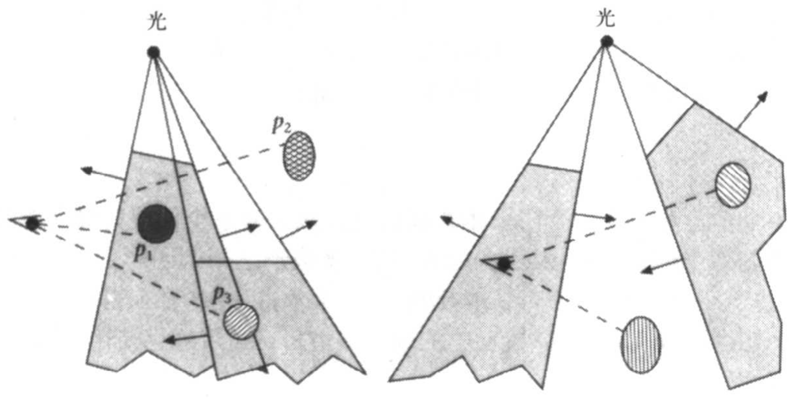


图14-4 点p位于阴影中，如果v与前向阴影平面相交多于与背向阴影平面的相交

阴影平面的产生可以预先处理，因为它是独立于视点的，而且只需执行一次，然后可以重复使用，只要对象不发生改变。阴影平面包含在场景数据库中，然后被送进图形管道。因为这些平面的多边形表示是必须的，它们的边界是由视域或光的影响范围裁剪所得。

对于第二步，在 Bergeron（1986）中渲染的执行是采用类似于第13章中所使用的扫描线算法。阴影平面的处理采用和可见场景多边形同样的方式处理。当它们出现在扫描线上的时候，不是用它们来设定像素颜色，而是用它们来增加/减少像素的阴影平面计数。我们在开始的时候将计数器设置为0，或者设置为包含视点的阴影体的数目。如果视点在阴影中，我们在渲染期间每当穿过一个前向阴影平面SP，就对计数器加1，每当穿过一个背向阴影平面SP，就对计数器减1。

如果我们使用OpenGL，那么第二个步骤可以通过硬件z缓冲区和模板缓冲区来实现（Heidmann，1991）。（*模板缓冲区是一个位图——虽然它在一些硬件上要复杂得多——这样位图中每一个比特位所对应的像素只有当比特位上的测试成功时才被改变。在 z测试成功的情形下，在模板缓冲区中的对应比特位将被修改。*）这是经过三个步骤完成的。第一步是针对打

开的光源对场景做渲染，不考虑阴影问题；然后是对阴影平面的渲染。在这个步骤中，我们允许与z缓冲区比较，但是每当比较成功，我们不是修改颜色缓冲区，而是增加/减少在模板缓冲区中对应位置的值。在这个渲染过程结束的时候模板缓冲区将包含每个像素的前向阴影平面数与背向阴影平面数之差；第三步对关闭的光照执行场景渲染，并使用模板缓冲区来限制
303 渲染范围，只渲染拥有更多前向SP的区域。

如果场景是由封闭多面体所构成的，那么对每个多边形单独地创建SP是非常浪费的，因为对于每个非轮廓边来说，我们将得到两个SP：一个为前向SP，一个为背向SP，它们会彼此删除对方。如果先计算从光源角度看时对象的框架并用这些边来代替，那将是更有效率的。无论如何，当场景具有一定复杂度的时候阴影平面的生成都是一个耗时的过程。另一个相关的计算阴影平面的方法是由McCool（2000）提出的。场景首先从光源位置进行绘制并读z缓冲区，然后基于离散化阴影图的边来重建阴影体。但是这个离散化过程导入了一些人为现象。

阴影体方法的缺点之一是阴影平面往往是非常大的（事实上它们是半无限的），而且它们对渲染时间是有负面效果的。实际上，我们可以限制这种效果，方法是不去求完全解而是只针对那些“重要”对象求阴影。通过创造并增加这些对象的阴影体到场景中以便进行阴影计算。就像阴影z缓冲区方法一样，我们可以使用这个方法将阴影投射到任何可以被扫描转换的对象。但与阴影z缓冲区不同的是，引起阴影的对象必须为多边形。多重光源可以通过创造单独的阴影体并对每个光源保留一个单独计数器的方法处理。

阴影体BSP树

对场景中每个多边形给定一个阴影体，我们能直接在对象空间中计算阴影并储存它们到数据库中，而不是像上面那样存储阴影平面。两个多边形之间的阴影，一个为遮光板（*O*），一个为接收器（*R*），可以通过*O*的阴影体对*R*的裁剪来求得。*R*的任何落在阴影体中的部分都在阴影中。这个阴影可以表示为在*R*表面上的细节多边形，或者通过使用阴影边拆分*R*为发亮部分和阴影部分。

在对象空间中执行计算的算法是要将每一个面向光源的多边形，比如说有n个这样的多边形，与每一个其他这种多边形的阴影体进行比较。然而这是浪费的。对这种n^2算法的改进可以根据这样的思想，即只有相对于光源较近的多边形才能投射阴影到较远的多边形上。多边形的排序可以通过构造场景BSP树并沿着相对于光源位置由前到后的方式遍历的方法。然后多边形可以以这个顺序处理，而且每一个只需要与位于它前面的SV做比较。这个方法能减少比较次数，如果由BSP树所生成的拆分数目不是太大。但是要获得更好的性能可以通过空间细分方法达到，如我们将会很快看到的那样。

304 在第11章中我们曾讨论用于排序多边形的BSP树数据结构。然而，BSP树的更大用处还是在于把它作为一种将空间细分为子空间或区域的层次化分解手段。每个节点的平面不只是将多边形拆分为两个子集——前和后（也可能在其上面），而且它也同时将空间拆分为两个子空间——正和负。然后每个孩子进一步地递归分离对应的子空间（Thibault and Naylor，1987）。在树的根节点上，我们所面对的是整个R^3空间（或R^2空间，或其他任何我们定义的场景空间，因为BSP树在任何维度空间中使用方法都是一样的），而在叶节点上所面对的是个别的凸区域。对于阴影体，我们可以标记这些区域为内部或外部，这要依据它们是否对应于空间中被照射的区域或位于阴影体区域。见图14-3b中的例子。

Chin和Feiner（1989）发展了这个思想，建立了一个对于所有面向光源的多边形的统一阴影体，即阴影体 BSP 树。我们将通过图14-5和图14-6中的例子来解释这个方法。

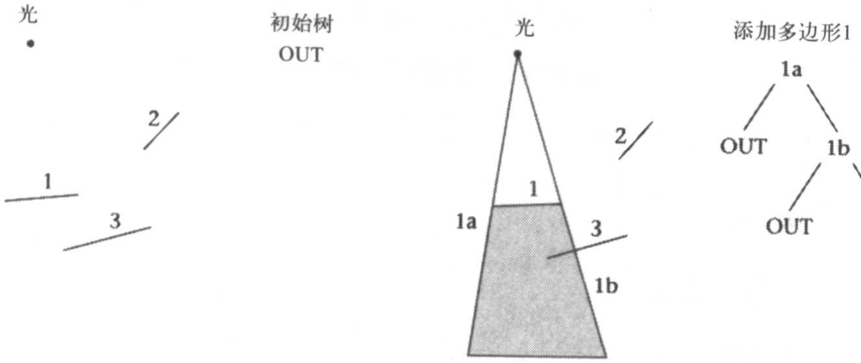


图14-5 建造SVBSP树：初始状态（左），添加过第一个多边形后的状态（右）

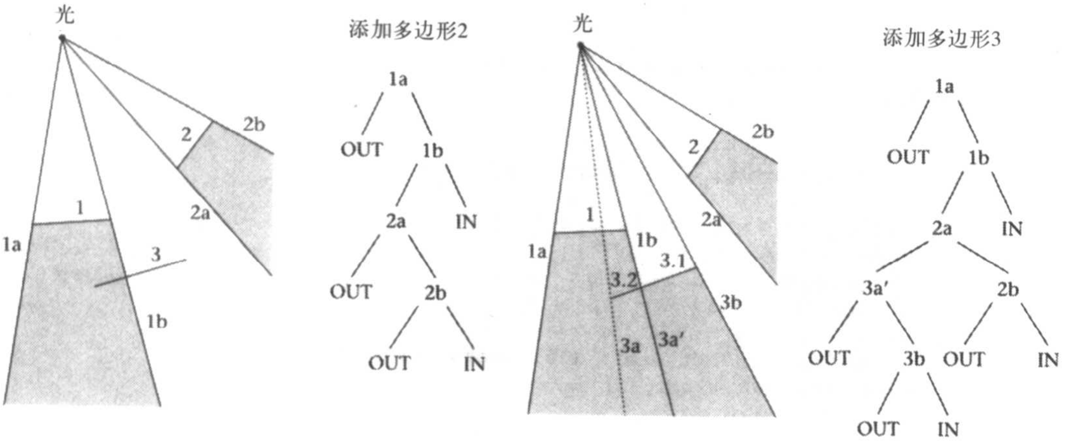


图14-6 建造SVBSP树：多边形2到达一个OUT单元使树增大（左）。
多边形3被拆分，与3.2一起到达一个IN区域

假设有一组多边形S和一个光源L，我们按照下面的步骤进行。首先建造一个BSP树，并 305
从光源的观察位置来获得一个由前到后的多边形排序。然后依次取这些多边形的每一个并把它们加入到 SVBSP 树。多边形的插入过程可以使用在第11章中所用过的渐增式构造 BSP 树的过程。在每个子树的根上多边形与平面做比较来确定是否将它插入在前面还是后面，或是拆分它，并将每块送到对应的子树中。这里的差别是在到达叶节点的时候。

如果叶节点标记为OUT，那么树不增加多边形平面，而是增加由那个片段的边所定义的阴影平面。多边形本身被划归为“被照射”。我们可以在图14-5 中看到这一点。最初 SVBSP 树只是一个OUT节点（左）。多边形 1 被插入到OUT节点中，因此它的阴影体用来替换那个节点。对于图14-6 中的多边形2（左）的情况是一样的。当我们从树的顶端插入它的时候，它是位于 SP 1a 之后的，所以被送到后面的子树中；在那里发现它是在SP 1b的前面，并结束于OUT节点，在这个节点上它用阴影体替换。

任何到达IN节点的片段都归类为被遮挡，但不改变树。图14-6（右）中的多边形 3在插入树中时被拆分，其中一个片段（3.2）结束于IN节点。

多边形本身不需要包括进树中，因为处理的顺序保证了它们将会总是位于那些已经在树中的

多边形的后面。另一个方法是以任意顺序处理场景多边形。它的好处是不需要 BSP 树排序，因此避免了增加场景多边形的数目。在这种情况下，多边形必须与每个阴影体一起包含在 SVBSP 树中。如果渲染中使用了z缓冲区，那么这个方法通常会执行得更快（Chrysanthou，1996）。

程序14-1　建造SVBSP树

```
buildSVBSP(Tree bsp, Light light)
{
    /* the svbsp tree is initially set to null */
    svbsp = OUT;

    /* the BSP gives the back-to-front order */
306 order[] = traverseBSP(bsp, light);

    /* each polygon is inserted in order */
    for(i = 0; i<=n;++i){
      svbsp = insert(svbsp, order[i], light);
    }

    /* tree is not needed any more, discard */
    free(svbsp);
}

SVBSP insert(SVBSP svbsp, Poly poly, Light L)
{
    if (leaf(svbsp)){ /* svbsp is a cell */
      if (svbsp == IN) {
          /* polygon in IN leaf, in shadow */
        add poly as a shadow polygon;
      } else
          /* polygon in OUT leaf, lit. expand tree*/
        return constructSV(poly, L);
      }
    } else
        /* find which side of the root is polygon */
        classifyPolygon(svbsp.rootplane, poly, pf, pb);
      if (notNull(pf)) {
        svbsp.front = insert(svbsp.front, pf, L);
      }
      if (notNull(pb)) {
        svbsp.back = insert(svbsp.back, pb, L);
      }
    }

    return svbsp;
}
```

这个过程的伪代码如程序14-1所示。在函数buildSVBSP中，SVBSP 首先被初始化为单个OUT节点，然后多边形通过使用场景 BSP 树得到排序，并按照这个顺序插入到 SVBSP 中。多边形的插入是使用递归函数insert。在每次对函数insert调用时，如果树是一个叶节点，那么如果它的值为IN，则将多边形标记为遮蔽；如果它的值为OUT，则叶节点由多边形的阴影体替换。如果树是一个内部节点，多边形根据在树的根节点的平面做分类并送到适当的子树中。在这个过程结束时树将被删除。

我们需要注意到的是，这个过程所包含的计算与初始BSP树的构造所需要的计算是一样的。这当然也需要通过平面对多边形做拆分。由此可见，如果我们有了构造 BSP 树所必须的

软件工具，阴影生成算法可以很容易构造出来。

这里仍然存在一个问题，那就是应该如何在数据结构中表示阴影。这可以通过两种方式完成。第一种方式是允许场景多边形在它们插入进SVBSP树的时候被拆分成“被照射”和“被遮挡”两种片段。这看起来是显然要做的事情，虽然将导致增加大量多边形，这些多边形的增加是因为每个最初的场景多边形根据每个落在其上的阴影被拆分为多个照射面片。另一种方法是保持最初的多边形完整并将阴影当作与每个初始多边形关联的细节多边形存储。此时场景多边形可以用完全光照渲染，并且阴影多边形将会覆盖适当的区域。阴影多边形可以渲染成只有环境光照的不透明物或者是透明的灰色。这里所增加的渲染多边形数目比较小，但是它们覆盖了一个较大的区域，因为阴影中的区域被有效地渲染了两次。

多重光源可以直接合并到这个算法中。对每个光源要执行一遍。第一个光源如上面方式处理。在此之后，使用第二个光源的位置来遍历最初的 BSP 树，以获得从这个光源的角度看 307
时由前到后顺序的多边形列表。这些多边形的处理完全与第一个光源的情况一样。现在惟一的附加需求是当遇到每个目标多边形的时候，它上面所记录的阴影（细节阴影多边形的列表）也必须当作目标来考虑。因为这些是普通的多边形，在这个过程中计算它们的时候，在它们上面也同样会记录有一个阴影列表。必须小心处理以保证只有当阴影多边形是由别的光源产生而不是当前处理的这个光源产生的时候才将该阴影多边形当作目标。因此如果我们使用第一个明暗处理方法，则每一个多边形也储存了那些已经被删除且不再对它的颜色起作用的那些光源集合。对于最初的场景多边形，这些集合将是空的。如果我们使用的是第二种方法，那么每个多边形储存的都是确实对它有贡献的光源集合。

在渲染过程期间，与场景多边形关联的阴影多边形要以一个正确的顺序显示，这样才能得到正确的阴影效果。首先，最初的场景多边形必须被显示。其次，所有的第一层阴影多边形必须被显示。所谓第一层的阴影多边形是那些直接来自于场景多边形与不同光源所对应的阴影体之间的交所形成的多边形。其次，所有的第二层阴影多边形必须被显示。这些是通过阴影体与第一层多边形相交所产生的多边形。

这个过程用图14-7来说明。该图给出了三个光源分别投射阴影到一个多边形上，所投射到的多边形标记为 0。阴影分别被标识为1、2和 3。我们假设光的处理顺序为1、2和 3。首先，

308

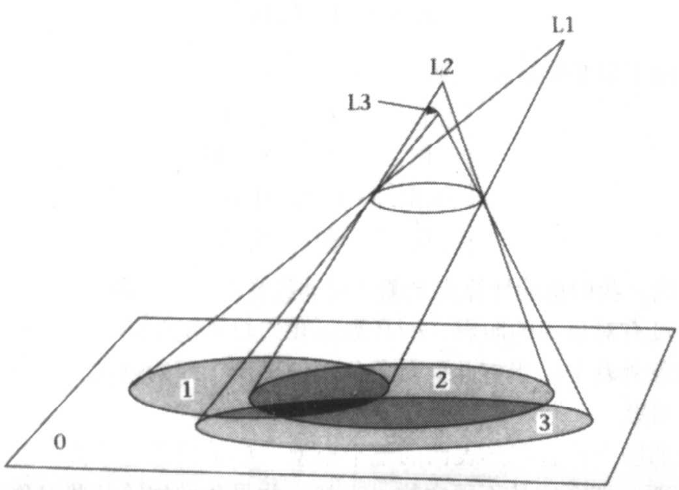


图14-7　多重光源的阴影

光源L1得到处理，产生阴影1。然后，L2得到处理，产生阴影 2 和 1∩2。最后 L3得到处理，产生阴影3、1∩3、2∩3和 1∩2∩3。当显示多边形 0 的时候，它是按照图14-7中所示的层次顺序进行显示的，即0、1、2、3、1∩2、1∩3、2∩3、1∩2∩3。

对 SVBSP 树的详细分析以及它与其他方法的比较可参见（Slater，1992a）。

树是只用于创造阴影，然后它就被丢弃掉。只要场景保持静态，阴影就可以在每一帧之间重复使用，无需重新计算。Chrysanthou和Slater（1995）保留了这种树结构，并说明了如何在场景发生很小改变的情况下有效地更新树。

伪阴影

这或许是一种在场景中创造阴影印象最为简单和最为快速的方法了，虽然它不是完全的也不总是准确的。这个方法首先是由 Blinn（1988）提出来的。阴影仅仅是在地平面上对最初多边形的投影。这里我们假设地面的平面方程为 $z=0$，但是也可以把它一般化为任何平面。这个方法的吸引力在于它可以使用一般常用的渲染图形管道来完成。在显示完没有阴影的场景之后，我们仅仅应用一个附加的变换矩阵将环境压平在地面上并重新渲染每个物体，让z缓冲区处理最后图像的合成。

假设有一个线光源，它发出的光线平行于方向 L（x_l、y_l、z_l），在场景中有一个点p（x_p、y_p、z_p）。那么我们可以求得p在地面上的阴影g，通过沿着光的方向对它做投影。投影将有形式（x_g，y_g，0）。我们需要定义一个矩阵 S，满足$p \cdot S=g$。

g位于由点 p和光的方向所定义的方程中：

$$g = p + t \cdot L$$

但是我们知道 $z_g=0$，因此可以求得 t为：

$$0 = z_p + t \cdot z_l$$

$$t = -z_p / z_l$$

所以求得：

$$x_g = x_p - (z_p / z_l) x_l$$

$$x_g = y_p - (z_p / z_l) y_l$$

这可以通过使用下列变换矩阵达到：

309

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -x_l / z_l & -y_l / z_l & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

沿着相同的路线，我们也能计算点光源（而非线光源）投射阴影的矩阵（见图14-8）。

在这个方法中没有对象间的阴影，而且如果我们想要避免阴影多边形穿越边的话，需要做相对于平面边的额外裁剪。当有几何对象位于地平面下面的时候，一个朴实的实现也将从这些对象向上投射阴影。

我们需要渲染那些上面有阴影表面的整个模型。如果只是地平面，那将是非常快的。如果想要更复杂的东西，那么它是不现实的。当然，如果我们知道某些对象在模型中占有重要的视觉地位，那么就可以决定只投影这些来创建阴影，从而极大地加速了这个方法。

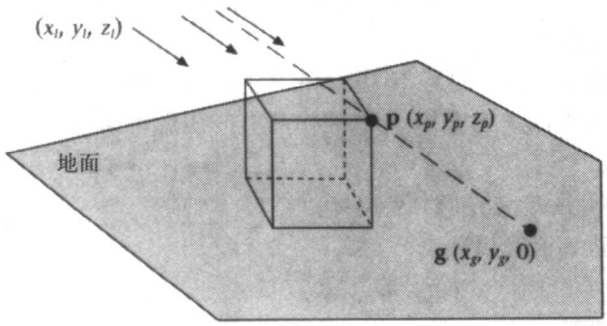


图14-8　将顶点投影到地平面上

14.3 阴影半影

来自点光源的阴影为我们提供了场景中对象间空间关系的很多信息。在真实世界中绝大部分光源都是具有一定面积的发光体。

为了增加图像的真实感，应该对这样的光源效果进行建模。面光源所产生的阴影有柔和的边，它们不再由一个硬边界（本影）所定义，而是还有部分光照射到的区域（半影）。在这一小节的稍后部分中我们将对本影和半影给出比较精确的定义。

关于面光源的阴影算法可以归结为两个较大的范畴：

解析确定：这一类方法解析地计算阴影的边界和半影里的其他不连续性。有些算法还在光照之前计算光源的可见部分。除了 Amanatides（1984）从圆形或球形的光源计算阴影之外，在这个种类中其他的方法都将模型限制为完全是由平面多边形所构成，包括光源。计算在对象空间中执行。 310

采样：这些是近似解法。通常将光源看成是一组点光源的集合，阴影为这些点光源的综合效果。光源的可见部分可以视为从某个给定位置所能看到的点光源比例。点采样技术的计算代价会是昂贵的，尤其是在需要一个精确解的时候，但是它们往往比解析确定方法更普遍。用现代的图形硬件实现采样技术是可能的，这样就使它们进一步走向了实用。我们还要把回旋方法放入这一类中，尽管它们不完全对光源进行点采样，但它们提供的是近似解。这些方法似乎比点采样方法更正确和更快速。

我们将从解析方法开始讲述。我们准备介绍原理并详细介绍其中一些方法，例如不连续网格化等。最后再简要介绍一些采样方法。

解析确定

在这些方法中，阴影和场景表面上光照函数的其他不连续性计算是通过显式构造“阴影平面”并跟踪它们到场景中来完成的。不久将会看到一个比“阴影平面”更恰当的名字。我们将在后面看到，如果我们想要精确地计算完全解，应该结合连续遮光板的贡献，这样这些“平面”实际上是二次曲面，但是还是让我们从比较简单的情况开始。我们将从本影和半影的边界是由单个遮光板引起的情形开始讨论。

极值阴影边界

来自面光源的阴影由一个完全封闭的区域（即所谓的本影）和一个部分封闭的区域（即

所谓的半影）所组成。在被照射到的区域和半影区域之间，以及在半影和本影区域之间的边界我们称之为阴影的极值边界。

首先提出计算精确极值边界的是 Nishita和 Nakamae（1983）。他们给出了如何通过极值平面来构造半影体和本影体（如图14-9）。假定所有的阴影平面都是面朝外的，远离阴影体，那么这些极值平面是:

对于半影：平面由一对（光源顶点，遮光板边）或者（光源边，遮光板顶点）定义，这里光源完全位于前半空间中，而遮光板完全位于后半空间中（如图14-9）。

对于本影：平面由光源顶点和遮光板的一条边定义，这里光源和遮光板都位于它们的后半空间中（如图14-9）。

311 那么，本影阴影体是本影平面的后半空间（负）与多边形的后半空间的交。同样半影阴影体是半影平面的后半空间和多边形的后半空间的交。

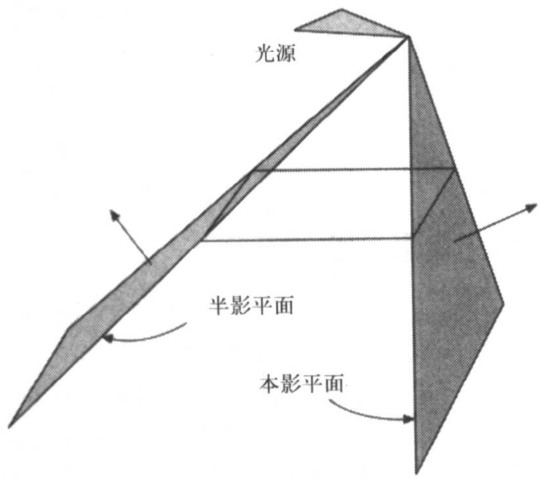


图14-9 极值阴影平面

在这一方法中，在场景多边形上的阴影边界以及本影和半影是在对象空间中计算的。这是通过将每个接收器多边形与每个其他多边形的半影阴影体做比较来完成的，如果有相交接收器也要与多边形的本影体做比较。很显然，我们所说的“每个其他多边形”只是指那些朝向光源或至少一定程度上朝向光源，而且至少一部分位于光源的前面的多边形（我们称这些多边形为*朝向光源的多边形*)。任何位于光源后面或者是其后面有光源的多边形都与任何阴影算法无关的。

312 一旦边界被求出来，它们就被转换到图像空间中，在此空间中它们将用于渲染时对多边形的光照。虽然图像是扫描转换到屏幕上的，光强度的计算无论何时只要遇到阴影边界就执行，对于扫描线的其他地方是以固定间隔执行。未被遮挡点的强度计算是应用式（14-1）进行的，如下所示。对于本影中的点这个值是零，只使用环境光强度。对于在半影中的点要对光源相对于该点的可见部分使用式（14-1)。

为了求出相对于半影点p的光源的可见部分，我们首先要求出对点的遮光板集合(O)，其次对O中的每个多边形构造一个棱锥体，该棱锥体以多边形的边为边，锥体的顶点位于p点。这与前面所描述的点阴影体是非常相似的，只是p点作了光源。光源与棱锥体做比较，只有落在棱锥体外面的部分才是可见的。这些可见的部分进而要与在O集合中的其余多边形的棱锥体做比较。

阴影边界和光照强度不是显式地存储在对象空间中的，所以无论何时视点发生了改变，整个过程必须被重复执行。而且，阴影边界确定是一个$O(n^2)$过程，这里n是朝向光源的场景多边形的数目。

Campbell 和 Fussell（1991）提出了一个更高效的算法，即在对象空间中执行所有的计算。所有朝向光源的多边形的半影和本影的阴影体被构造为 BSP 树，然后将它们组合在一起形成两个 SVBSP 树：一个针对半影，另一个针对本影。SVBSP树的构造是通过BSP树的阴影体以及由 Naylor等（1990）给出的算法。然后每一个朝向光源的多边形插入到半影树中以便确定

阴影边界。如果我们发现它部分位于半影 SVBSP 的标记为IN的单元中，那么还要将它与本影树做比较测试。在点光源的 SVBSP方法中（Chin and Feiner，1989），多边形的分类可以在构造树的过程中插入多边形的时候进行。这里算法使用了两遍过程，第一遍是构造 SVBSP树，第二遍是向树中插入多边形。这是必须的，因为不像点光源的情况，求多边形相对于光源的顺序不是一件容易的事情。

Chin和 Feiner（1992）处理针对某一个面光源的排序问题是通过对光源进行拆分来完成的，每当发现某个光源跨骑在一个场景多边形的平面上的时候，就拆分它。于是场景 BSP 树能从每个结果光源片段开始遍历以得到由前到后的排序。对每个源片段构造两个 SVBSP 树，而且在构造期间以类似于点SVBSP 树的方式计算阴影。

这两种方法中没有一种方法能准确掌握是哪一个多边形遮挡了在半影顶点上的光源，对于光照计算这是需要的。Campbell 和 Fussell 为每个接收器和光源构造凸包，并用它来裁剪所有的场景多边形。在接收器上的半影顶点只需要测试相对于剩余多边形的光源。Chin和 Feiner 使用 BSP 树来求多边形的集合(O)，即位于光源和接收器之间的多边形集合。对于每个位于接收器上的半影顶点v_i，我们利用多边形集合O构造一个点 SVBSP 树，v_i作为顶端。这样光源插入到这棵树中且可见部分就是那些到达标记为OUT单元的部分。

我们把从半影顶点处观察到的光源可见部分当作凸多边形计算，这可以当作单独的光源来看待，这些光源共同决定了在那个顶点上的总体光照。

来自于凸多边形光源的光照I_p可以用下列等式来描述，如 Nishita and Nakamae（1983）中所描述的。这里假设凸多边形光源在点p有n个顶点：

$$I_p = \frac{I_s}{2}\sum_{v=1}^{n}\theta_v \cos(\phi_v) \tag{14-1}$$ 313

这里，

I_s =光源的光强度；

θ_v= 由光源顶点v、p和光源顶点v+1所构成的角度，如图14-10；

ϕ_v=p点所在平面与三角形 v、p 和v+1之间的角度，如图14-10。

在所有上面所提到的方法中，只有本影和半影边界的确定是明确的，但是事实上光照函数有局部最大值、最小值、以及在半影区域里面的不连续性（Heckbert，1991；Campbell and Fussell，1991）。

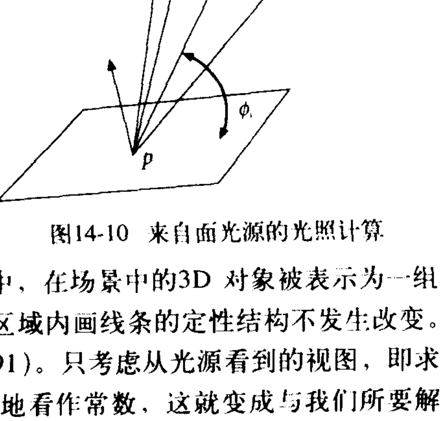


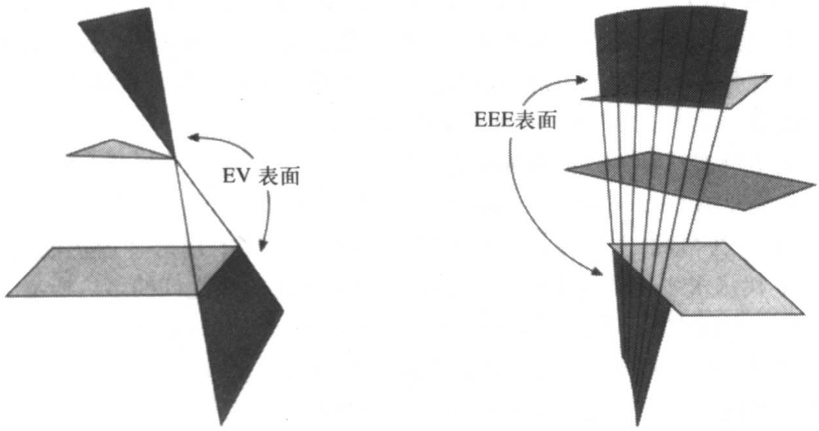
图14-10 来自面光源的光照计算

特征图

通过一种被称为特征图的方法，可以对这个问题在半影里的变化有一个更深刻的认识。所谓的特征图方法通常用在计算机视觉对象识别应用中。在这个方法中，在场景中的3D 对象被表示为一组二维视图，视点空间被分割成一些区域，这样在每一个区域内画线条的定性结构不发生改变。对每个图像结构的定性度量称为特征（Gigus et al.，1991）。只考虑从光源看到的视图，即求出这样的空间区域，在这个区域中光源的可见部分定性地看作常数，这就变成与我们所要解决的问题一样的问题了。

在特征图理论中，包围这些均匀区域的表面称为*关键表面*，在穿过它们的时候产生*视觉*
314 *事件*。关键表面是通过场景中边和顶点的交互来定义的，如 Gigus 在Gigus and Malik（1990）中所描述的，它们可以被分为两类。

- EV*表面*：由边和顶点所定义的平面，如图14-11a所示。
- EEE*表面*：由三个不相邻的边所定义的二次曲面，如图14-11b所示。



a）由边和顶点所定义的EV表面　　b）由三个不相邻的边所定义的EEE表面

图　14-11

对于阴影计算问题这些表面大部分是不相关的。我们只对EV和EEE表面感兴趣，这些表面包含光源特征（边或顶点），或者是切分了光源表面（Drettakis，1994）。这些表面与场景多边形的交产生关键曲线，这些关键曲线对应于光照函数中的不连续性。这些关键曲线也被称为不连续性曲线（或EV事件的边）。

半影体如前面所定义的，完全由EV表面所构成，EV表面包含了光源的特征，而本影体可能由EV表面和EEE表面所构成。所有的不连续性都封闭在半影中。

不连续网格化

我们称函数 f 是在区间（t_1，t_2）上连续的（C^0），当且仅当

$$\forall a \in (t_1,t_2), \lim_{x\to a-\varepsilon} f(x) = \lim_{x\to a+\varepsilon} f(x) = f(a), \varepsilon \to 0 \tag{14-2}$$

不满足这一条件的函数我们称之为零阶(D^0)不连续。一个函数的k阶导数满足式（14-2），我们就说它是C^k连续的。一个函数若是 C^{k-1} 但不是C^k 的，则我们称它是D^k的。

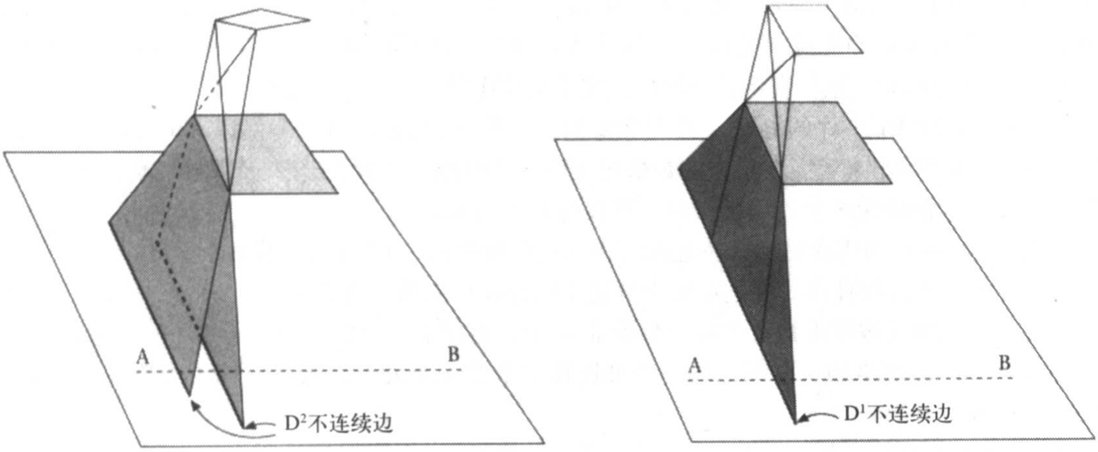
315 在多边形的光照函数中不连续性是由它与关键表面的交引起的。如前所述，惟一相关的关键事件是由那些包含光源边或顶点的EV表面和 EEE 表面所引起的，这些事件被称为*发射器事件*，那些不包含光源特征的EV表面或 EEE 表面，但是它们的表面与光源相交的事件，被称为*非发射器事件*。

正如 Heckbert（1991）所描述的，由点光源、线光源和面光源引起的关键表面通常具有D^0、D^1和 D^2，但是当两个不连续性发生重合的时候，不连续性的阶就要下降。

因为我们正在处理面光源，由EV和 EEE 表面所造成的不连续性一般会是二阶不连续性。举例来说，在图14-12a中，当我们在多边形上沿着直线AB向前移动的时候，光照函数在与有

标记的边交叉的点上具有D^2不连续性。

对于光源的边和遮光板的边相平行的情况，两个EV表面具有一个组合效果，产生D^1边。在图14-12b中，如果我们在多边形上沿着直线 AB 移动，光照函数在与有标记的边交叉的点上具有 D^1 不连续性。



a）二阶（D^2）不连续性　　b）D^1不连续性

图 14-12

D^0边也同样会产生。这些发生在表面之间接触的时候，如果没有被显式地表示的话，它们能引起某些最严重的假象。发现它们通常需要在执行进一步细分之前对数据库专门执行一遍处理（Baum et al.，1991）（见图14-13）。

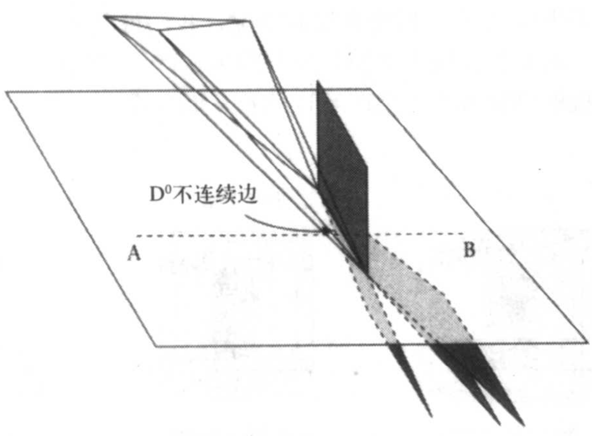


图14-13 D^0不连续性

当光源的光强度不均匀的时候，高阶不连续性就可能发生。通常光源上的D^k能引起 接收器上的D^{k+1}和 D^{k+2}（Heckbert，1991）。高阶不连续性很少得到注意，通常凡是高于D^2的我们都不考虑。

Heckbert（1992a）首先在2D范围内对不连续网格化进行了研究。完全的网格是通过分析场景中边和顶点之间的每个可能的交互来构造的，这是一个复杂度为N^3的操作，N为顶点数。 316
他后来又将工作扩展到 3D 环境 （Heclbert，1992b），使用的是相似的算法，即跟踪每一个EV表面，而EEE表面被忽略。与此同时，Lishinski 等（1992）提出了一个不同的3D算法，他

们也只考虑了EV不连续性，但是使用了一个更"渐进式的"方式去定位它们。对每个发光多边形单独计算。在每一遍中选择最高能量的多边形作为光源，求出由它所引起的其他多边形上的不连续性并计算强度。在最后，所产生的网格合并在一起以便形成最后的细分。这个方法在后面的 DM 研究中得到采用。

EEE表面由Teller（1992）处理了一部分，他是在一个相关计算中提出来的，这个计算是求经过一系列入口的光源可见区域。这个方法求出了本影的极值边界，但是算法是基于5D Plucker坐标表示的，这种表示很难推广到完全的DM 解，而且计算复杂度高。

包括EV和EEE事件的算法，甚至非发射器，后来都被 Drettakis和Fiume（1994）以及Stewart和Ghali提出来了。在绝大多数情况下EEE表面和非发射器EV表面都被忽略掉了，这是因为由它们的例外所产生的误差相对于其代价是很小的。

按照Lishinski渐进式算法，不连续网格化可以被分解为4步主要的操作：

（1）求不连续性曲线，通过在整个环境中跟踪发射器的关键表面。

（2）这些曲线被用来构造在每一个场景多边形/面片上的网格，针对于特殊的发射器。

（3）计算在网格的每个顶点和其他被选择点上的光照强度。强度计算需要确定从每个点看去光源的可见部分。

317 这3个步骤对每一个主要的发射器重复执行，最后：

（4）创建在每个表面上的网格，并合并它们来形成最后的子划分。

这最后一步只有对多重光源（例如为得到辐射度解）才是重要的，因此我们不准备在这里描述它。在绝大多数算法中，定位不连续性和构造网格的任务是交替进行的，但是我们在这里对它们分开讨论。

在图14-14中我们看到的是阴影和不连续性的例子，这是从不同光源/接收器几何体投射到接收器上形成的。对于矩形光源/矩形遮光板的情况，我们可以从图14-15中看到。在图14-14中，黑色区域是本影，而灰色区域为半影。所有的边是D^2不连续性的，这符合我们前面的讨论，因为遮光板没有接触接收器而产生D^0不连续性，而且没有一条光源边与遮光板的边平行而产生D^1不连续性。

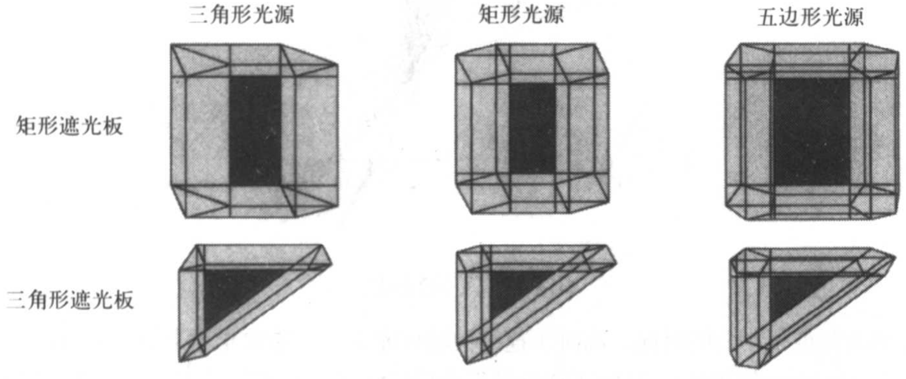


图14-14　不同几何体对的阴影和不连续性

不连续性的定位

绝大多数的 DM 算法定位 D^0 边、D^1边和D^2边所使用的是不同的方法。D^0不连续性是位于

两个相互接触表面的相交点上的，对它的计算首先只考虑对象的接近度。这包括访问每个对象并将它与毗连的一些对象进行比较。这个运算的效率依赖于确定接近方法的效率。Tampieri（1993）使用包围体的层次结构，而Drettakis 和Fiume（1994）使用一个基于体素的细分结构。 318

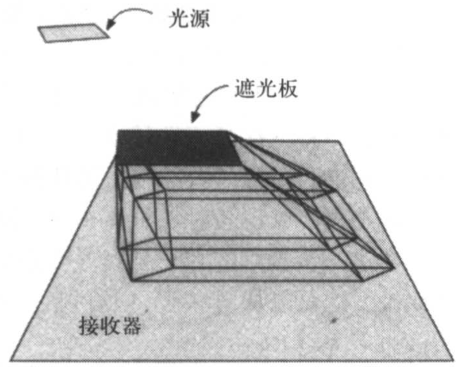
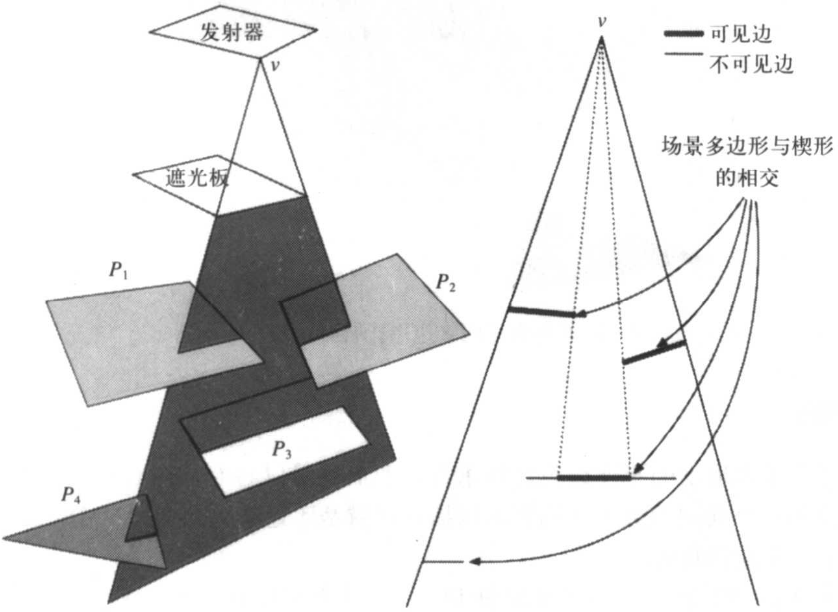


图14-15 先前图像的光源/接收器的摆放

为求出其余的不连续性，即如果忽略EEE或非发射器EV时EV发射器边，最通常的方法是构成一个半无限的楔形，这个楔形是利用光源的顶点和遮光板的边构成的，或者是利用光源的一条边和遮光板的顶点构成的，并求出它们与场景多边形的交。在其余部分的讨论中，我们将区分由光源顶点所引起的楔形和由光源边所构成的楔形。这是通过调用前者的 VE和后者的EV楔形完成的。EV楔形和它在环境中的一些相交如图14-16a所示。

这些算法单独处理每个EV楔形来求出与环境的交，但是它们的主要区别在于它们是否对场景多边形排序以及是否用相同的顺序去比较楔形和这些多边形。在后者这一组中我们有Heckbert和 Drettakis两个代表人物。Heckbert将每个场景多边形与每个楔形做比较，需要额外的计算。Drettakis 极大地减少了比较的次数，他通过使用一个基于体素的细分方法，该方法限制候选多边形为那些只与楔形共享体素的多边形。

以无序方式处理多边形的一个问题是，不可能马上知道是否多边形与楔形之间的相交是不连续（关键边）。只有相交边从楔形的顶点处看去是可见的时候才构成不连续边，并应该添加到网格中。比如，在图14-16a中多边形P_4的交点不能从顶点v处看到，因为它被更靠近v的多边形遮挡了，由此它不应该被添加到网格中来。 319



a）楔形与场景多边形相交　b）将相交转换到楔形空间来确定可见性

图 14-16

为了从每个楔形中找出关键边，我们需要将相交点转换到楔形平面并且放入排序的列表中。从楔形顶点的角度执行二维可见性测试，这个测试是Weiler-Atherton 可见表面算法（Atherton et al.，1978；Weiler and Atherton，1977）的一个变种，于是我们就可以求出关键边。在图14-16b中只有粗边对应的是不连续性。

另外一种方法是将场景多边形构造到一棵BSP 树中，这个BSP树将提供从每个楔形顶点角度的顺序，排除了对二维可见性测试的需要（Lischinski et al.，1992）。当多边形以由前到后顺序与楔形做比较时，求出相交，同时楔形被相交的多边形裁剪，这样 只有不被遮挡的部分才会进一步被跟踪。一旦楔形被完全裁剪，跟踪就马上停止，避免了不必要的多边形/楔形的比较。这可以从图14-17 中看到：那些未形成不连续的相交没有被发现，对楔形的跟踪停止于多边形 P_3。

Chrysanthou（1996）给出了另外一个方法。将对应于一个遮光板的EV楔形集合当做一
320 个实体（一个阴影体），而且它们在场景中是一起被跟踪的。阴影体与接收器多边形的候选列表进行比较，这个候选列表是使用基于盖瓦立方体的方法得到的。D^0是在相同的一遍处理中求出的。

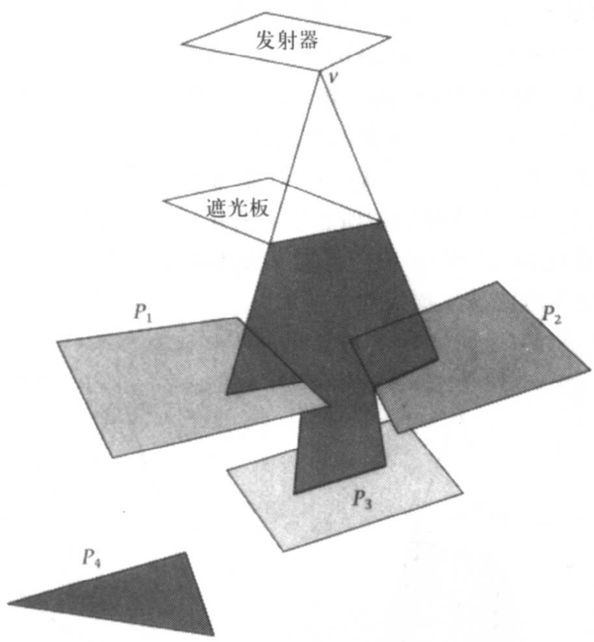


图14-17 在与有序的多边形比较时对楔形的裁剪

面上的网格构造

一旦在每个多边形上的不连续边已经求得，我们就可以将它们结合起来形成多边形上的网格。需要使用一个共同的数据结构表示网格，这就是不连续**网格树**（DM树）。对每一个场景多边形使用一个这样的树。

DM 树由两部分组成：一个二维BSP树和一个翼边数据结构（WEDS）（Baumgart，1974，1975）。WEDS是基于边的结构，适合于维护一致性并加速存取邻接信息，如同我们在先前所看到的。它有三个基本元素：顶点、边和面结构。绝大多数拓扑信息是存在于边结构中的。

边结构有指针，分别指向它所在的两个面，指向它的两个端点，以及指向四个其他与其共享顶点的边。每个顶点结构保存一个3D点以及数个指针，这些指针分别是指向其所在边的指针，而每个面结构保存有指向定义面边界的那些边的指针。

翼边结构是 DM 树的核心，这是因为它为每个顶点的强度只计算一次和在关联面之间共享创造了条件。同时它保证了在面拆分时不引入 T顶点。

二维BSP树是第11章中所描述的结构的增强版本。与先前一样，每个内部节点保存一个子超平面（边），且它是通过超平面（线）定义的。每个节点对应于一个空间区域，这个空间区域将被叶节点进一步细分，叶节点对应于一个不再细分的区域（单元或网格面）。

然而，在这种情况下我们在叶节点上储存的是二维空间区域的一个显式表示，通过网格面上的指针保持二维空间区域与叶节点的对应关系。

最初DM 树只有一个叶节点，其中保存了一个面（整个多边形），如图14-18a所示。当有一个不连续边添加进来的时候，它拆分了多边形，于是树得到更新，在根节点上存储这个边，其两个新的面构成了它的两个叶子。如果边不能完全地跨越面，那么就对它增加另外的一段来扩展它，所形成的新边称为**构造边**，由此来保持细分为凸的（图14-18b）。当有更多的不连续边被添加进来时，沿着DM树对它们进行过滤，可能在途中被细分直到到达叶节点，在叶节点上它们细分该节点上的面。

这个网格构造方法的潜在问题之一是，在正确的点上连接不连续边依赖于机器精确性。举例来说，在图14-18 中的两条边e_1和e_2。这两条边是由遮光板的连续边（或者是连续顶点） 321
构成的楔形所产生的，这就是为什么它们共享一个共同端点（v）的原因。当每个楔形被独立地跟踪而且不连续性是一个一个被插入结构中的时候，如果在计算中没有精度误差发生，它们只会正确地连接于v点。这是一个共同的问题，发生在许多应用中，而且通常都是通过利用容错值给直线增加厚度来解决的。当然，如果我们有非常小的边或者是密集放置的边，还会产生其他的问题。

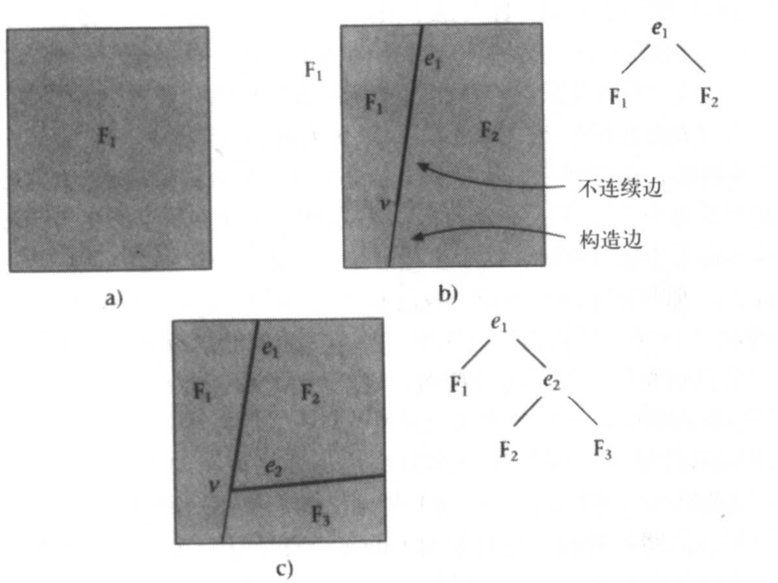


图14-18 建造DM树

Drettakis（1994）的工作是采取一些方法来对这个问题进行限制。创建EV楔形并以遮光板的边顺序追踪，并将同一个接收器上的每个新不连续边连接到先前那个的尾部。之所以可能，是因为使用扩展的WEDS来适应临时的悬挂边，而无需使用二维BSP 树。然而这只能正确地连接边总数的一半，ve事件仍然依赖于机器的精度。Chrysanthou（1996）提出的算法处理所有的不连续性，这些不连续性是从单个遮光板投射到接收器平面上的，并构成单独的 DM树。这棵树然后被合并到接收器总的DM树之中，用这种方式这样的误差被进一步减到最小。

用一棵 BSP 树存储不连续性（DM树）不是没有问题。它能产生带有不好的特征比的网格元素，而且在前面插入树中的不连续性会对网格的其他不连续性带来负面影响。Heckbert（1992b）使用了有约束的 Delauney三角化，而Drettakis（1994） 提出了使用一般网格作为被插入的不连续性所在的基础网格。

网格顶点的光照

如上所创造的网格的每个顶点（包括在进一步的网格细化过程如三角化过程中所产生的所有顶点）都必须有光照。网格的光照计算通常是DM代价最大的部分。这是因为对于每个顶
322 点，光源的可见部分必须在对它们应用式（14-1）之前求出。求相对于一个顶点（v）光源（S）可见部分的一般方法是将v的遮光板投影到光源平面，然后使用它们去裁剪掉光源的任何隐藏部分。当然，只有在半影中的顶点需要做光源/ 遮光板比较，但是，如果已知顶点是其中一个多边形的顶点的话，绝大部分DM方法不能够提供直接信息关于哪个顶点在半影中或者哪个多边形引起的半影。

把场景中的每个多边形都作为一个可能的遮光板，这种做法是非常低效的。Lischinski 等（1992）通过使用杆状剔除技术来限制可能的遮光板数目（Haines and Wallace，1991）。对于网格中的每个顶点v，我们构造一个棱锥体，使得v成为棱锥体的光源。场景多边形与这个棱锥体进行比较，只有那些与棱锥体内部相交的多边形才被投影到光源平面以便裁剪光源。

Gatenby（1995）使用空间相关性对每个顶点提供一个比较小的可能遮光板集合。它所依赖的事实是，如果遮光板O相对于光源不遮挡给定接收器R的任何部分，那么在R上没有顶点会在O的半影中，因而O应该从顶点的可能遮光板集合中排除。在对接收器多边形R的网格顶点光照之前，对整个R要进行两步"预处理"，以便求出那些至少影响它的部分的多边形。第一遍是对BSP树的遍历，由前到后，从每一个光源顶点开始直到发现R。由遍历所找到的多边形集合合在一起形成集合 L_0'，然后将接收器与在 L_0' 中的每个多边形的半影阴影体做比较。当发现R与 L_0' 中的多边形半影有交时，这个多边形被添加到另一个列表 L_0中。R中网格顶点的可能遮光板是集合 L_0。那些完全落在本影中或没有被遮挡的接收器因此能特别快速地处理。

在Chrysanthou（1996）所描述的算法中，每个区域的遮光板标识符在构造网格的时候插入到网格元素，所以在光照时间不需要搜索。这是可能的，因为来自每个遮光板的不连续性全都被合并到接收器的树中，这样允许从一开始就对不同的区域分类。

一种完全不同的计算光源可见部分的方法是基于特征图的方法，分别由Drettakis 和Fiume（1994）以及Stewart和Ghali（1994）提出。他们使用一种称为反向投影的数据结构，它存储光源可见部分的精确结构。这对表面上的每个点计算一次，然后每当跨越不连续边到达附近单元的时候就渐增地更新它。与其他方法相比它是快速的，缺点是需要构造一个完整的网格，该网格包括非发射器EV和 EEE 边。

在彩图14-19中我们所看到的左边的图像是由Drettakis的反向投影方法生成的，右图为不连续性。 323

实践中的不连续网格化

不连续网格化方法能产生高度正确的解。然而，它们通常会遇到伸缩性和鲁棒性问题。这主要是因为表面上所得的不连续网格化存在太多的边。在复杂场景中大量不连续性对最后图像的影响是很小的。尤其是当场景受到多于一个光源照射的时候，因为一个光源的光照会冲刷掉来自另外一个光源的不连续性。这可以用图14-20 很好地加以说明。输入场景由5742个多边形所组成，但是在网格中生成了近500 000条不连续性直线和超过4百万个三角形元素。这是一个极端的例子，因为光照来自于天花板上的24个条状光源，而且还有特殊的几何体布置。然而，它清楚地表明我们要善于选择。

关于如何淘汰那些对视觉效果作用很小的不连续性有许多的建议。Tampieri（1993）在每条边上获得一些采样，如果它的强度根据辐射度量度低于一个阈值，那么它不被包含在内。Hardt 和Teller（1996）也使用辐射度量度。Hedley（1998）另一方面提出基于视觉感知量度的方法，它看起来比其他方法更好。

计算一个DM 解是相当耗时的：它可能需要花上数分钟，甚至几个小时。它可以被预先处理然后使用于漫游应用中。然而，对于那些只有一小部分几何体发生改变的动态环境来说，当我们与一个小的对象进行交互的时候，再一次计算解是浪费的。

在前一小节中所提到的一些方法已经扩展成适应于动态修改。Worrall和同事（Worrall et al.，1995；Worrall et al.，1998）建立了一个三角化不连续网格化，然后当投射对象移动的时 324
候，从一个表面到另一个表面定位不连续边。他们在处理过程中提出了减少多边形的多种技术。他们用光线投射于新的网格顶点来计算光照，这是代价很高的。

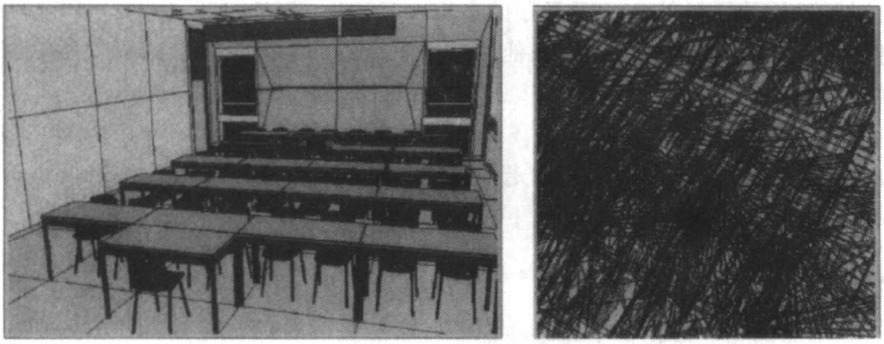


图14-20 不连续网格化的复杂度。左边是有5742个多边形的输入场景，右边是对某个桌子下面地板的近距离观察，显示了500 000多条不连续性直线（由Bristol大学计算机科学系的David Hedley提供）

Chrysanthou 和Slater（1997）利用修改的半立方体快速地识别那些在动画中会受到影响的有关对象（见第15章）。他们也使用BSP树合并来精确求解对于每个新的网格顶点什么是遮光板。这个方法像是有前途的，但它只对少于200个多边形的场景做过测试，据此做出评估是困难的。最后 Loscos 和 Drettakis（1997）扩展了 Drettakis（1994）方法。他们在对象移动时使用空间相关性来局部化事件中的变化。使用一个运动体去选择3D网格的体素，在这个网格

中信息是被预先储存好的。更新算法使用 Worral 算法的一个扩展算法来求出可见性修改，并更新在不连续表面和对象之间的交。通过对EV不连续网格之间的相交的分析来检测什么时候EEE 表面应该被建立、维护和销毁。然而，它们仍然需要有一个完整的网格以便通过反向投影计算光照。

采样

在这一小节中我们看到的方法不可能提供如上所述的精度水平，但是它们十分简单，而且较容易实现。它们之中许多都利用专门化的图形硬件，随着这些专业化的硬件逐渐变得更快和更普遍，我们期待着这些方法也变得更流行。

大部分点光源阴影方法可以被扩展作为模拟柔和阴影的方法，这是通过对近似于面光源的一簇点光源重复地应用这种方法来完成的。举例来说，光线跟踪方法可以被自然地扩展来做这项工作，通过朝向光源投射多个采样阴影光线，而不是一个。在分布式光线跟踪中，这些点被散开使得走样减到最少（Cook et al.，1984）。半立方体或者是在辐射度中使用的相似方法也可以在这里介绍，但是我们准备在下一章中对它们进行更详细的研究。

325 在本章开始处所看到的阴影缓冲区方法可以被扩展到柔和阴影，这需要使用硬件支持的深度缓冲区（Haeberli and Akeley，1990）。场景经过多遍渲染，使用面光源上的不同点，使用硬件集聚缓冲区对结果求平均（有关集聚缓冲区及其应用的描述请见第23.6节）。这个方法需要很多采样光源，以便得到好的阴影质量，但是这可以得到加速，通过只计算少数采样和应用视图插值生成中间阴影（Chen and Williams，1993）。Herf 和 Heckbert（1996）也使用了集聚缓冲区：创建许多阴影图像，每一个对应于光源上的一个采样点；然后注册它们并在接收器上求平均，并结合使用集聚缓冲区；然后将结果存储为即将应用于接收器的纹理。

Brotman和Badler（1984）使用由面光源上的一些样本点所产生的阴影体，然后在每个像素上使用扩展的z缓冲区算法和链表来存储多重阴影体所需的信息。Heidmann（1991）也使用了阴影体，但他是用硬件实现的。在他的方法中场景的渲染是通过从每个光源创建的模板缓冲区完成的。

最后柔和阴影也被计算出来，使用在图像上回旋（Soler and Sillion，1998）的方法，这是在对从点光源的阴影做平滑处理或反走样处理（Reeves et al.，1987），且使用基于图像的技术（Keating and Max，1999）的情况下实现。

14.4 小结

在这一章中我们在一定程度上研究了阴影确定问题。我们始终是在“实时”模式中进行讨论的。虽然阴影本影和半影被详细地处理了，但仍然没有对象间的反射，因此没有全局光
326 照。阴影半影较之阴影本影显得更“真实”，但是它们仍然是不够真实的。在下一章中我们将转到全局光照方法。正如光线跟踪自然地计算阴影本影，但是不能够轻松地处理半影，辐射度方法恰恰相反——它能自然地计算阴影半影，但是在区分本影硬边界时存在相当的困难。尽管如此，在这一章中讨论的许多技术对辐射度计算有直接的应用，同时也在计算机图形渲染的很多其他方面有直接的应用。

第15章 辐射度介绍

15.1 引言

在光线跟踪（第6章）中我们介绍的光照模型和算法能够充分地处理光在场景各处的镜面反射和传导。它是一个全局光照模型。对于镜面传导的光，它处理光源和对象之间的交互以及对象之间的交互。它没有正确地考虑到漫反射光传送的全局问题，而是使用固定的“环境光”项作为代替。对于许多现实的场景（举例来说，现在环顾你的四周），绝大部分场景中的光能量是来自漫反射表面的，反之只有一小部分场景具有镜面高光、反射和折射（传导）。这可能是为什么许多光线跟踪图像尽管很壮观，却总是不能反映日常的场景，而是表现各种不同的对象（尤其是球体）之间反射的原因。

辐射度光照模型考虑漫反射的全局问题但不对镜面反射建模。它是基于热力工程师所使 327
用的方法来计算在封闭环境中表面之间的辐射能量的交换。这个方法首先是由Goral等人（1984）引入到计算机图形学中来的。

在这一章中我们将对辐射度作基本介绍。这也将是“综合性”的一章，因为它把本书的两个主题紧密结合在一起：实时方面（辐射度对于全局光照场景的快速漫游需要Gouraud明暗处理），它是一种全局光照方法，同时在计算的关键方面也利用了光线投射（第 5 章）。

15.2 形状因子：两面片之间的能量

辐射度模式提供一个全局光照模型，考虑到封闭漫反射环境中光线的相互反射，也就是说，环境里的所有光能量是来自于环境本身的光源，没有光逃离出环境。我们假设场景描述为许多的表面（举例来说多边形），而且这些表面被进一步细分为小的微分区域，称之为*面片*。

我们在第3章中曾经将光能量用“辐射能量”或“通量”来度量——即每单位时间中流过真实或想象表面的能量（单位为瓦特）。***辐射度***（radiosity）是离开表面上每单位面积的辐射能量（单位面积上的瓦特数（w/m^2））。***辐照度***（irradiance）是一种相似的度量，只是它所表示的是表面上每单位面积入射通量。辐射度和辐照度都是与波长相关的——它们的值应该分别对所有的波长进行计算，但是我们在本章余下部分的讨论中不准备明确涉及波长问题。

我们首先考虑两面片间的能量关系，即光源面片S和接收器面片R之间的能量关系，为了方便叙述我们重新将图3.2标记为这里的图15-1。假设从光源到接收器的光亮度（radiance）是L，因此由式（3-10）我们有：

$$\mathrm{d}\Phi = L \cdot \mathrm{d}S \cdot \cos\theta_s \cdot \mathrm{d}\omega_R \qquad (15\text{-}1)$$

这里Φ是从S到R的光通量。展开微分立体角，使用式（2-28）：

$$\mathrm{d}\Phi = \frac{L \cdot \mathrm{d}S \cdot \cos\theta_s \cdot \mathrm{d}R \cdot \cos\theta_R}{r^2} \qquad (15\text{-}2)$$

我们现在考虑从S出发的所有方向上的总的光通量（Φ）。使用式（15-2）有：

328

$$\Phi = \int_{\Omega} L(p,\omega)\,\mathrm{d}S\cos\theta_S\,\mathrm{d}\omega \tag{15-3}$$

这里Ω是S上对应于光照半球的所有方向的集合，其中心位于p点。现在我们假设环境只由漫反射表面组成。

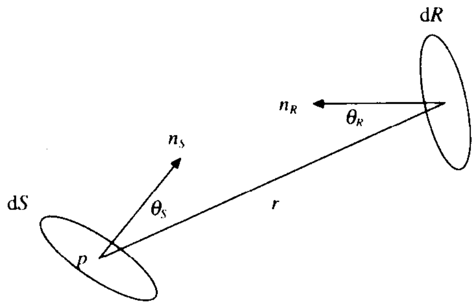


图15-1 两面片间的辐射能量

此时$L(p,\ \omega)=L$，因为光亮度在所有的方向上是一样的，而且我们为简便起见不再讨论p。我们使用式（2-26），有：

$$\begin{aligned}\Phi &= L\mathrm{d}S\int_0^{2\pi}\int_0^{\pi/2}\cos\theta\sin\theta\mathrm{d}\theta\mathrm{d}\phi \\ &= L\mathrm{d}S\pi\end{aligned} \tag{15-4}$$

从式（15-2）和式（15-4）中我们得到，在R的每单位面积上所接收到的来自光源S的光通量比例是：

$$F_{\mathrm{d}S,\mathrm{d}R} = \frac{\mathrm{d}\Phi}{\Phi} = \frac{\cos\theta_S\cdot\cos\theta_R}{\pi r^2} \tag{15-5}$$

这个量被称为从S到R的微分形状因子。它是一个能量的比例数（用辐射能量或光通量表示），指的是来自于光源的微分面片而到达接收器微分面片的能量，是以接收器上的每个单位面积进行计算的。

如果重复相同的参数，只是这次从微分区域dS到面片R考虑能量的比例，我们获得：

$$F_{\mathrm{d}S,R} = \int_R \frac{\cos\theta_S\cdot\cos\theta_R}{\pi r^2}\mathrm{d}R \tag{15-6}$$

最后如果我们整合所有光源面片，那么光源每单位面积的能量到达接收面片的能量比例是：

$$F_{S,R} = \frac{1}{A_S}\int_S\int_R \frac{\cos\theta_S\cdot\cos\theta_R}{\pi r^2}\mathrm{d}R\,\mathrm{d}S \tag{15-7}$$

这是从光源面片S到接收器面片R的形状因子。

通常我们用面片i和j分别表示A_i和A_j，其对应的角度为θ_i和θ_j。那么从i到j的形状因子，即每单位光源面积和接收机单位面积离开i到达j的能量比例是：

329

$$F_{ij} = \frac{1}{A_i}\cdot\int_{A_i}\int_{A_j} \frac{\cos\theta_i\cdot\cos\theta_j}{\pi r^2}\mathrm{d}A_j\,\mathrm{d}A_i \tag{15-8}$$

注意我们从这可以得出对称关系:

$$A_i F_{ij} = A_j F_{ji} \tag{15-9}$$

形状因子在下面将要描述的辐射度算法中扮演着一个十分重要的角色。

15.3 辐射度方程

既然我们假定是在封闭环境中，离开一个面片的辐射能量一定等于直接从面片发出的功率（如果它本身就是一个光源）加上从该面片的总的反射。这与导出光亮度方程式（3-23）的结论相同，只是现在的特殊上下文是，环境由许多的漫反射面片所组成（i=1，2，…，n）。如果我们将离开第i个面片的光通量记为Φ_i，那么:

$$\Phi_i = \Phi_{ei} + \rho_i \sum_j \Phi_j F_{ji} \tag{15-10}$$

这里Φ_{ei}是从面片i所发出的光通量，ρ_i是i的反射率（由i接收并反射出去的光通量的比例）。代数和是来自所有的其他面片j乘以相应的形状因子所得的光通量——所以只有从j到达i的实际总数计算在内。

这个方程表达为光通量的形式。为了将它转换成辐射度，我们要利用辐射度的定义，即辐射度是每单位面积的光通量。因此如果B_i是每个面片i的辐射度，A_i为面积，那么$\Phi_i = B_i A_i$，因此有:

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{ji} \tag{15-11}$$

这里E_i是从i发出的辐射度。

使用式（15-9）我们得到:

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_i F_{ij} \tag{15-12}$$

或

$$B_i = E_i + \rho_i \sum_j B_j F_{ij} \tag{15-13}$$

这个方程的含义（当然这个方程对每个波长可以有一个拷贝）是相当直观的。它说明来自面片i的辐射度等于从i直接发出的辐射度，加上它对入射的辐照度反射的比例。进入的能量 330
是接收来自其他每一个面片（j）的辐照度的总和，但是要减去来自这个面片且到达i的能量比例（由形状因子确定）。

假设除辐射度外的其他所有项都是已知的，我们就得到了一个由n个线性方程n个变量构成的系统，这可以表达为矩阵形式:

$$FB=E \tag{15-14}$$

这里:

$$F = \begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \dots & \dots & \dots & \dots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1-\rho_n F_{nn} \end{bmatrix} \tag{15-15}$$

$$B=\begin{bmatrix}B_1\\B_2\\\cdots\\B_n\end{bmatrix},E=\begin{bmatrix}E_1\\E_2\\\cdots\\E_n\end{bmatrix} \tag{15-16}$$

对B求解方程组得到所需的辐射度解。如果真的能得到所需要的所有各个量，那就方便了。不幸的是，实际情况并不是这样的。

这个方法需要E和ρ项是已知的。这些决定于表面所表示的材料性质，是由场景设计者选择的（同样注意这些依赖于波长的量）。更加棘手的问题是形状因子的计算。

15.4　形状因子的计算

该方法的主要组成部分需要形状因子F_{ij}被计算出来。形状因子是一个纯粹的几何量，独立于在场景中真实的光照条件。如果一个对象被其他对象遮挡或者是部分地被遮挡，那么有一部分光能量不能达到对象的表面，这一点并没有得到式（15-5）~（15-8）的考虑。

实际上，形状因子是不能够解析地计算的，而且无论何时都需要将对象之间的可见性关系考虑进去。在最初的关于辐射度的文章中（Goral et al.，1984），积分表达式（15-8）通过使用Stoke定理被转换成一个复杂的围线积分，然后用数值方法沿着轮廓线求积分。这是不实用的，即使是对最简单的场景。

Nusselt类比和半立方体逼近

在比较新的一篇文章中（Cohen and Greenberg，1985）用到一个结果，该结果被称为Nusselt类比。它说明形状因子可以这样计算：通过在一个面片上放置一个半球，投影另一个
331 面片到这个半球上，然后再垂直地投影到半球底面上的圆面上。我们得到投影所占据圆形区域的一小部分等于差分区域到有限区域的形状因子 $F_{dA_iA_j}$ ，这可以通过图15-2 说明。

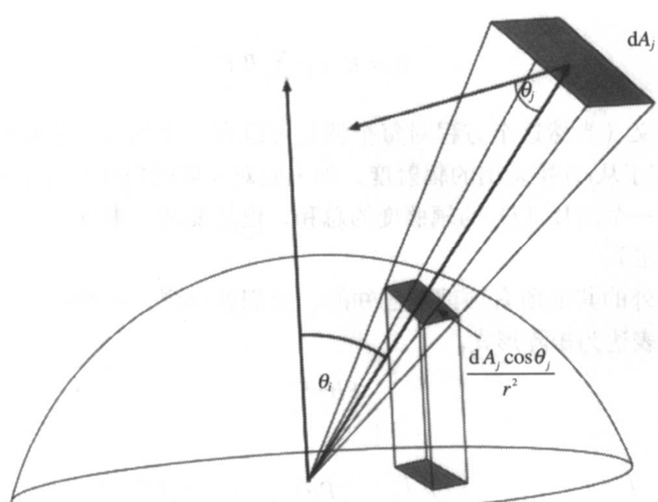


图15-2　Nusselt类比

这个结果可以被直观地显示出来。假设面片A_j的总面积被分割成许多微分区域dA_j，如前

所述。那么微分区域到半球面的投影是dA_j所对应的立体角:

$$\frac{dA_j\cos\theta_j}{r^2} \tag{15-17}$$

这里r是到dA_j的距离。因此它在圆形区域上的投影所占据的投影区域是:

$$\left(\frac{dA_j\cos\theta_j}{r^2}\right)\cos\theta_i \tag{15-18}$$

这个投影所占据的圆形区域的比例是:

$$\left(\frac{dA_j\cos\theta_j}{\pi r^2}\right)\cos\theta_i \tag{15-19}$$

现在在这个面片的区域上积分，得到 $F_{dA_iA_j}$ 的结果，等价于式（15-6）。

图15-2也说明了另一个重要结论，即那些在半球上有相同投影的所有面片关于微分区域 dA_i有相同的形状因子。基于这一点，Cohen 和 Greenberg（1985）设计了一个实用的形状因子计算方法。我们假设面片是足够小的，这样形状因子 $F_{A_iA_j}$ 可以用 $F_{dA_iA_j}$ 近似。

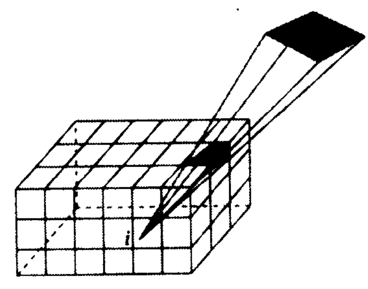


图15-3 用半立方体计算形状因子

332

图15-3 给出了一个面片i，一个半立方体从它的中心点建立起来。多边形j是面片i在半立方体上的投影，以便确定从j到“微分区域”i的形状因子。环境中的每个多边形以这种方式投影到i，使用隐藏表面算法确定哪个多边形是从i可见的。半立方体的表面被铺以盖瓦，投影到这个表面的多边形的标识符储存在盖瓦中。可以建立一个查找表，给出该表面的盖瓦对面片中心的贡献结果。举例来说，容易说明半立方体表面上的一个面片的贡献为:

$$\frac{1}{\pi(x^2+y^2+1)^2} \tag{15-20}$$

这些被称为“delta形状因子”。最后的形状因子是特定面片的delta形状因子的总和，因此它的计算是在覆盖的盖瓦集合上对如式（15-20）中这样的项求和。

人们提出用z缓冲区作为对半立方体表面上的隐藏表面消去手段，这里“像素”对应于立方体的盖瓦。因此每个面是铺以盖瓦的，只有当在关联的z缓冲区中的对应入口被允许的时候面片的表示符才能写入盖瓦。使用这个方法所有的面片必须被投影，然后盖瓦必须被遍历，以便计算delta形状因子对各种不同的面片的贡献。

另一种方法是使用 BSP 树。它的工作过程是这样的:

- 形成最初多边形（不是多边形上的面片）的 BSP 树。
- 设半立方体底面的中心是COP，从这个位置开始对树遍历，构成多边形由前到后顺序的多边形链表。
- 对这个链表中第i个多边形，将它的每个面片投影到半立方体上。每块盖瓦只需要一个布尔值，当此盖瓦被任何面片达到的时候该布尔值为真，其初始设定为假。一旦为真，它便无法被任何其他面片使用。 333

在这个方法中，面片的所有delta形状因子是在投影的时候求出的（而不是投影所有的面片，然后确定形状因子，如z缓冲区那样）。

用光线投射计算形状因子

现在我们掉转方向，并应用第5章的一些思想来计算形状因子。Wallace等（1989）介绍了一种基于光线投射计算形状因子的方法。不像半立方体方法，计算表面一个小的面片上的辐射度，然后平均这些顶点以便进行平滑的明暗处理，光线投射方法直接计算顶点上的辐射度。与一表面面片的微分区域关联的形状因子由式（15-6）给出。

这可以通过对A_j的delta小区域求代数和（数值积分）求得。当A_j与面片之间的距离r相比很大的时候这个数值是不稳定的。相应地，新的方法依赖于基于已知解析解的近似值。从微分区 dA到垂直相对的半径为a的接收圆盘的形状因子由下式给出：

$$F_{\mathrm{d}A_i A} = \frac{a^2}{r^2 + a^2} \tag{15-21}$$

由形状因子的互反性原则有：

$$\mathrm{d}A_i F_{\mathrm{d}A_i A_j} = A_j F_{A_j \mathrm{d}A_i} \tag{15-22}$$

所以从作为光源的圆盘到作为接收器的微分区域的形状因子由下式给出：

$$F_{A_j \mathrm{d}A_i} = \frac{\mathrm{d}A_j}{\pi r^2 + A_j} \tag{15-23}$$

该式成立，因为$A_j=\pi a^2$，即圆盘的面积。

这个关系是对于垂直相对的圆盘。更一般化地，包括每个表面的法向和光源与接收器之间方向的角度余弦。

$$F_{A_j \mathrm{d}A_i} = \frac{\cos\theta_i \cos\theta_j \,\mathrm{d}A_j}{\pi r^2 + A_j} \tag{15-24}$$

附加的近似值假设区域是盘状的。

通过这个表达式求形状因子是需要光线投射来达到的。光源面片（A_j）被分解为一些微分区域，接收器（dA_i）是一个特殊的顶点。从顶点开始跟踪光线到场景中特别的delta区域，如果光线在到达光源中心之前停止，我们就得到了隐藏表面遮挡。

这个顶点的最后形状因子等于从该顶点可见的delta形状因子的总和，如式（15-25）所示。

334

$$F_{A_j \mathrm{d}A_i} = \sum_{k=1}^{n} \mathrm{d}_i \cdot \frac{\cos(\theta_{ik})\cos(\theta_j)\,\mathrm{d}A_{ik}}{\pi r^2 + A_j} \tag{15-25}$$

这里n是光源上样本点的数目，d_i=0或1，且依赖于光源上样本点相对于该顶点来说是否可见。

这种计算形状因子的方法如我们所预期的，当光源区域的形状越接近圆盘的时候就越精确。举例来说，正方形能给出一个相当好的结果，而矩形就不能。相对于在半立方体方法中所需要的网格尺寸，通常我们只需要很少的样本点就能得到很好的效果。

因为辐射度已经在顶点上直接计算出来了，在渲染期间的插值阶段不需要从面片的中心转换到顶点（见15.7节“渲染”）。这是这种方法的另外一个优势。

15.5　渐进细化方法

算法

这个计算辐射度的方法是由Cohen等人（1988）提出来的。在式（15-12）中代数和中每

个单项确定了自面片j到面片i的辐射度的贡献。

$$B_i \text{ due to } B_j = \rho_i B_j F_{ij} \tag{15-26}$$

因为有$F_{ij}A_i=F_{ji}A_j$（交换i和j）、我们有：

$$B_j \text{ due to } B_i = \rho_j B_i F_{ji} = \rho_j B_i F_{ij} \frac{A_i}{A_j} \tag{15-27}$$

因此对所有的面片j：B_j due to $B_i = \rho_j B_i F_{ij} \dfrac{A_i}{A_j}$

我们从面片i“发射”出光线，允许它对环境的所有其他面片j产生贡献，调整每一个其他面片的辐射度。这里重要的一点是在这个公式中所需要的形状因子（F_{ij}）仍然是那些基于在面片i处的半立方体。在每次迭代中将会有对B_i估计的一个改进，它会包括B_i的先前估计。因此只有B_i的改变需要考虑、这称为**未击中辐射度**ΔB_i。

最初我们将所有的非光源B_i和ΔB_i设为零，并将其他的部分都设为E_i。那么一个迭代过程从求最亮的面片开始并使用它射出能量。在每个步骤中重复它，所用的最明亮面片是具有最大的未击中能量的那个面片。当总的未击中能量低于一个阈值的时候这个过程结束，或者当结果已经满意过程即结束。

```
while not converged {
   Find patch i with the greatest AiΔBi;
   compute the form factors Fij using a hemi-cube at i;
   for each patch j {
      ΔRad = ρjΔBiFij(Ai/Aj);
      ΔBj = ΔBj + ΔRad;
      Bj = Bj + ΔRad;
   }
   ΔBi = 0;
   /*render scene here if desired*/
}
```

335

这个方法来自于这样的观测，即在一个场景中的全局光照主要取决于很少几个元素（主光源和少数几个次要光源）。因此根据其重要性顺序处理它们能迅速收敛到完全解。它同时避免了对完全的形状因子矩阵的二次方的储存需求，这是因为只有与当前面片有关的形状因子在每次迭代中被计算然后丢弃。

包括环境项

使用渐进细化方法容许在每次迭代之后渲染场景。既然面片是以辐射度降序排列的，光源将会首先被处理，这样一些光将会在第一次迭代之后增加到环境中。然而实际上，我们可以看到这些初始场景是黑暗的。因此纯粹为了显示的目的，我们需要给它一个启发式校正，即添加一个常数的环境项到光照中，这个项在每次迭代中逐渐变小。

Cohen 等（1998）提出从i到j的初始形状因子可以用面积比近似：

$$F_{ij} = \frac{A_j}{\sum_{i=1}^{n} A_i} \tag{15-28}$$

环境的平均反射率可以被计算为：

$$\rho_{ave}=\frac{\sum_{i=1}^{n}\rho_i A_i}{\sum_{i=1}^{n}A_i} \tag{15-29}$$

能量进入环境中的部分ρ_{ave}将会被反射回来，其中又有一部分将会反射出去，如此这般，这就导致了全部的平均相互反射R:

$$R=1+\rho_{ave}+\rho_{ave}^2+\rho_{ave}^3+\cdots=\frac{1}{1-\rho_{ave}} \tag{15-30}$$

总的环境辐射度等于辐射度的面积平均值乘以反射因子R，辐射度的面积平均值是经由形状因子未被“击中”的：

$$Ambient=R\sum_{j=1}^{n}\Delta B_j F_{*j} \tag{15-31}$$

因此为了显示目的只使用辐射度如下：

336

$$B_i'=B_i+\rho_i Ambient \tag{15-32}$$

15.6　网格化

实际上，在辐射度方法中我们试图通过离散化的做法捕获在场景表面上的光照变化，这个离散化过程是将每个面片离散化成一些足够小的面片网格，以便保证在每一个小的面片上光亮度分布处处是均匀的。这可以通过下面所讨论的多种方法来达到。

一致网格化

第一个辐射度方法是基于一致网格化的方法。这是一种最简单的实现。这里每个表面被分割成规则的矩形面片网格，并计算在每个矩形面片中心处的辐射度。网格的分辨率控制精度和求解速度。

获得每个表面的适当分辨率不是一件容易的任务，这总是要留给用户去完成。然而，这个方法是存在一些问题的，举例来说：

- 由于低采样频率，丢失了阴影或斑驳的阴影。
- 阴影漏洞和光线漏洞——这是更明显的，因为在光照中存在大梯度（例如下面所述的D^0不连续性）。

自适应细分

Cohen等（1986）提出了一种子结构化方法，这种方法和完全矩阵辐射度解配合使用。然而，它也能用于渐进细化。

我们需要网格足够精细以便捕获表面上的所有光照变化（阴影、高光区等）。但是我们也想要减小处理代价，因为这种代价是面片数目的二次方。这里对我们有帮助的一个观察是虽然我们需要小面片接收光照来捕获细节，但是还是可以使用比较大的面片发出辐射度。

子结构化就是基于这个思想。环境被初始分割成一些粗糙面片。这些将在整个求解过程中当作发射器。我们将进一步分割这些面片直到达到要求。

在完全矩阵解中子结构化是这样使用的。我们首先获得基于面片的一个完全解，然后通过细分网格来对整个网格求精。我们通过比较附近面片的辐射度，每当发现它们大于某个预先定义的阈值时，我们假设在辐射度上有一个重大的梯度，并进一步细分这些面片。 337

新单元上的辐射度是通过聚集来自环境的其他面片的辐射度计算得到的（而不是从单元本身）。

$$B_{i_e} = E_{i_e} + \rho_{i_e} \sum_j B_j \cdot F_{i_e j} \tag{15-33}$$

这里，B_{i_e}、E_{i_e}和ρ_{i_e}是面片i上单元e的辐射度、发射强度以及反射率，B_j是面片j的辐射度，$F_{i_e j}$是从单元i_e到面片j的形状因子。

B_j是已知的，$F_{i_e j}$的计算可以通过在单元上使用半立方体计算出来。一旦新单元上的辐射度被计算出来，面片上的辐射度就可以用一个更精确的估计值来更新它，并用于任何进一步的迭代：

$$B_i = \frac{1}{A_i} \sum_e B_{i_e} \cdot A_{i_e} \tag{15-34}$$

这里B_i和A_i分别是面片的辐射度和面片面积，B_{i_e}和A_{i_e}是面片i的单元e上的辐射度和单元e的面积。

那么我们可以连续进行更多的迭代，将需要拆分的单元继续拆分并进一步细化网格。如果我们假设面片都是矩形面片，那么通常是用每个面片上的四叉树数据结构存储细分结果。

如果我们有初始面片数目N和最后的单元数目M，那么在这个方法中计算$N \times M$个形状因子，这个数远小于$M \times M$，因为$M > N$。

对于渐进细化我们利用了一个相似的思想。我们使用面片发射辐射度。在每个渐进细化迭代中我们选择具有最大未击中能量的面片，并发射能量到环境中的单元上。当单元从这一步中所接收到的辐射度之间存在很大的差异时，细分它们并从当前发射面片执行一个新的发射。

层次化辐射度

层次化辐射度（HR）（Hanrahan et al.，1991）将上面的思想推进了一步。在子结构化中，能量的交换是从面片到单元，不去关心个别交互的重要性，而HR在每个面片上创建一个子面片的层次结构，并在能量会发生交换的那些细分层次上将每个子面片单独与其他每个子面片连接起来。

我们从一组非常粗糙的初始面片集合开始。我们对每一对尝试去“连接”它们以便进行能量交换。主要思想是如果在两者之间的形状因子（FF）是大的，那么在交换中就潜在着很大的误差。在这种情况下我们细分它们并检查每对子面片的FF。如果它们低于阈值，我们就连接它们；否则我们进一步细分，直到FF足够得低，或者子面片的面积足够小。 338

在这个过程中我们在每个面片上建造一个层次结构，比如一个四叉树，当然也可以是另外类型的层次结构。然后我们重新执行这个过程，尽量让它与其他面片成对。在这个连接中，部分层次结构可能已经存在，所以我们需要做的就是在适当的节点上增加连接并对需要细化的地方进行细分。能量的交换只发生在连接上。

在HR中，对于位于层次结构底部靠近叶节点的两个面片i、j，我们将要它们连接在一起，这样的连接上有细微的交换发生；而如果j离得很远，那么会有来自层次结构根节点的集中交

换。当然也会有中间的情况。

算法分两个阶段执行。第一个阶段是在每个初始面片上建造层次结构，同时建立所有的连接。然后是通过迭代过程沿着这些连接分布辐射度。

为了建立层次结构和连接，我们在每对初始的面片之间调用refine子程序。

```
void refine (p, q, F_eps, A_eps) {
   estimate F_pq and F_qp;
   if (F_pq < F_eps and F_qp < F_eps)
      link (p, q);
   else {
      if (F_pq > F_qp and A_p > A_eps)
         for each child c of p
            refine(p_c, q, F_eps, A_eps);
      else if (A_q > A_eps)
         for each child c of q
            refine(p, q_c, F_eps, A_eps);
      else
         link(p, q)
   }
}
```

这里，F_{eps}和A_{eps}给出两面片间的最大形状因子的边界以及子分割的最小面积。我们可以看看图15-4，这是在两个正交多边形上建造层次结构的例子。

连接过程的最大代价是形状因子的计算。这将会发生得非常频繁，所以在实际过程中不易精确地计算，而是代之以估计的方法。举例来说，我们可以使用下列公式：

$$F_{ij} = \frac{\cos\theta_i \cos\theta_j A_j}{\pi r^2 + A_j}$$

上面所说的并没有考虑面片之间的可见性。这对于真实环境是重要的。因为我们考虑两
339 个面片对之间的连接，而不是考虑一个面片和环境中其余部分之间的连接，半立方体方法会是浪费的。光线投射方法在这里是比较适当的。可以在面片对之间发射多条光线以便得到可见性的估计。这个计算可以在单纯基于FF的细分之后执行，此时我们使用可见性来调整向下连接的FF；或者它也可以在refine子程序执行期间去做，也就是当建立层次结构的时候做。在后一种情况中部分可见性可以被用来鼓励进一步的细分以便获得更好的阴影边界。

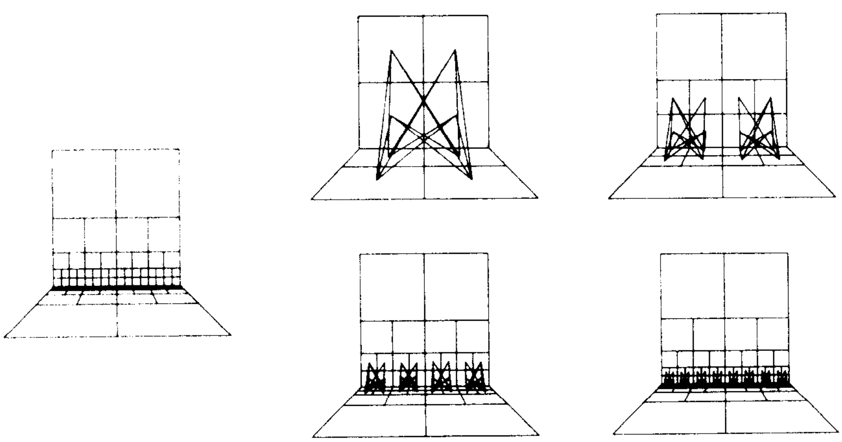


图15-4 两个正交多边形上的层次化细分。直线段显示层次结构中不同水平上的连接对

（由美国斯坦福大学计算机科学和电机工程系的Pat Hanrahan 提供）

一旦连接已经建立，我们就要着手通过迭代过程进行辐射度的传播。在每一步中，我们首先通过连接聚集面片上的辐射度。面片i到其他面片j的贡献可以用公式B_i+=$\rho_i F_{ij} B_j$来计算。在下面的伪代码中我们使用Bg来标记在这次迭代中由该面片所聚集的辐射度。

```
void gather (patch p) {
   Bg_p = 0;
   for each link of p connecting to patch j
      Bg_p += r_pF_pjB_j;
   for each child c of p
      gather (p_c)
}
```

然而，每个子面片只与场景的一个子集直接相连。为了表示到达区域的总能量我们需要考虑在层次结构中较高连接和较低连接上有什么样的聚集。在聚集之后，从树的顶端开始我们使用深度优先遍历对树层次结构进行遍历，为下一次迭代更新能量。所聚集的能量下推至叶节点，并对途中的内部面片增加能量。然后使用面积加权平均将它再一次拉到根节点，从而完成更新。

Hanrahan认为这个方法的复杂度可以达到$O(M)$，这里M是场景中最后单元的数目。为此我们需要付出面片的开始配对代价$O(N^2)$。初始面片是大得多的，所以也就比自适应细分方法中的多。 340

不连续网格化

虽然自适应或层次化技术通过对光照变化比较大的区域提供更多的单元来降低所得解的误差，但是它们仅仅是最小化误差，而不是像我们在"一致网格化"小节中所提到的对问题的全新处理。为了删除人为现象（如光/阴影漏洞等），我们需要构造网格以便包括由视觉事件所引起的光照不连续性。在第14章中我们描述了如何计算和添加它们到网格中。然而，我们是集中在单个光源的光照上。对于不连续网格化的辐射度，我们需要计算来自多重光源的不连续性并将它们加在一起（Lischinski et al.，1992；Drettakis and Fiume，1994）。结果是高精确的，但是该方法代价相当大，很难稳健地实现而且通常是过度的。一个更好的方法应该只选择这些不连续性的子集，而忽略掉那些小的分布（Gibson and Hubbold，1997）。

15.7 渲染

假设我们使用了上述方法之一，场景中所有面片的辐射度得到计算。如何渲染场景呢？让我们忽略掉关于辐射度值如何转换成显示器适当RGB值的复杂处理——这已经在第4章中的光亮度部分中讨论过了。假设现在这已经通过某些方法完成了。辐射度归属于面片的中心。如果我们天真地按照面片的辐射度用一致的颜色来渲染面片，那么所获得的将是一个有小面的图像——在邻接面片之间的边会显得很突出。我们在先前就已经遇到过这个问题。我们似乎是将辐射度当作与多边形相关联的"预定"颜色。在第13章中我们采用了插值平滑明暗处理方法来解决这个问题，特别是Gouraud 明暗处理使用于漫反射。

现在辐射度方法只模拟漫反射。因此Gouraud明暗处理会使用到。惟一的问题是对于Gouraud明暗处理来说，辐射度值一定要与顶点关联在一起，而不是与面片的中心关联。克服这一点的方法应该是常见的——在每个顶点上产生辐射度作为周围面片的平均辐射度。这与计算法向的近似值是相似的。它也是使用翼边数据结构的一个好理由。

15.8 小结

本章给出了由漫反射表面所组成场景的全局光照的简要介绍。该方法的主要计算代价是对形状因子的计算——正如相交计算的主要工作在于光线跟踪一样。

然而，这里有一点不同。辐射度方法是可以“脱线”执行的——它不是渲染过程本身的
341~342 一部分。一旦辐射度解已经获得，场景的实时漫游就是可能的了，因为只有 Gouraud 明暗处理和z缓冲区——也就是说标准图形硬件是渲染所需要的。这里要强调的一点是，因为辐射度模拟漫反射，而漫反射是视图独立的，所以辐射度计算可以独立于视图执行。然而，光线跟踪是对视图依赖性很强的——整个渲染过程开始于投影中心。

十分逼真的图像可以用辐射度来生成（见彩图P-2）。然而，辐射度方法只对已知场景几何描述和光照描述的情况是有效的。如果光照发生改变，那么至少形状因子无需重新计算，但是仍需要渐进细化来发射光线到场景中。另一方面，如果场景中任何对象的几何属性发生了改变，举例来说被平移了，那么计算必须被重新进行。Chen（1990）给出了在给定场景几何属性发生变化时如何渐进地计算变化。

辐射度和光线跟踪是互相补充的，因而需要整合在一起。已经有一些工作是针对这个问题的，尽管算法代价都非常高。两个例子分别参见Wallace et al.（1989）和Chen（1991）。

第16章 快速光线跟踪

16.1 引言

在前一章中我们看到了如何渲染一个全局光照的场景，假设所有的表面都是漫反射器。在某种程度上这符合实时全局光照的目标——因为我们至少能够实时地漫游这样的场景。然而，显而易见的一点是需要大量的预处理步骤来计算辐射度，而且一般来讲不可能实时地计算场景的变化。

在这一章中我们将回到光线跟踪。到目前为止光线跟踪方法还是一种十分“粗放”的方法，每一条光线都要与场景中每个对象进行相交测试。在这一章中我们将讨论一些能把光线跟踪速 343
度提高几个数量级的思想。不幸的是，这仍然与实时的要求有相当的差距，但是这些思想所带来的改进可以是从小时到分钟，或者是从数日到几小时不等，这依赖于场景的复杂度。

我们从相交计算问题开始，然后介绍几种有助于排除原本需要遍历的大量光线的数据结构，这些数据结构不会对最后的图像带来根本影响。这些数据结构很多不仅对光线跟踪有帮助，在其他方面也发挥着重要作用，因此它们本身也是很值得研究的。

16.2 相交计算

光线跟踪中绝大多数工作在于相交计算。使用最多的基本图形是多边形。我们在第8章中讨论了光线与多边形的相交问题。在第5章中我们看到了如何计算光线与球体相交，而在第18章中我们将要研究光线与更一般的对象的相交问题，这些对象即所谓的二次曲面，如球面和平面就是特例。

绝大部分有关光线跟踪的研究都集中在减少相交计算的计算量这个问题上，要么降低每条光线的代价，要么减少光线的总数，或者是两者兼而有之。

现有的加速光线可见性计算的算法是基于对空间相关性的利用。在可见性计算中只有那些穿越光线路径上区域的对象是需要考虑的；其他对象将不会与光线相交，因而可以被忽略掉。这些算法的主要问题是如何恰当地分割世界空间以便让被测试的对象数目减到最少，并减少遍历空间来寻找候选对象的开销。这些算法可以分为以下几个种类：

- 层次化包围体：这些包围体是通过对象的位置和尺寸确定的；
- 一致空间细分：构造固定而一致的空间分割，此种分割独立于对象的分布；
- 自适应空间细分：使用基于对象分布的一种自适应细分模式；
- 光线方向技术：光线之间的关联性诱导对象空间分类，这样相似的光线束可以被一起处理。

我们将依次考虑这些内容。

16.3 包围体和层次结构

包围体

减少相交代价的一个明显的方法是对每个基本体素对象安装一个包围体，例如一个球体

或一个方盒。这需要在包围体松紧度和光线与包围体相交计算的简单性之间采取一个折衷。如果光线不与包围体相交，那么就无需进一步的行动。然而，如果光线确实与包围体相交，此时需要进一步的测试以发现光线是否确实与所包围的对象相交。如果包围体不是将对象裹
344 得很紧的话，那就有很大可能出现光线与包围体相交但却不与所包围的对象相交的情况。在某些情况下会有比原先所做的更多的工作。包围体在光线跟踪中很少独自使用，一般都是用来形成层次结构。

层次化包围体

在这个方法中包围体是按体素做的，那么高层包围体包裹一组低层的包围体，如此等等（Gold-smith and Salmon，1987；Kay and Kajiya，1986），这由图16-1说明。这里有多个对象，分别标记为A到F，每个上面都有一个包围盒，将包围体成组地集合在一起从而形成包围体的层次结构。根节点包围体包裹的是整个场景。每条光线渐进地与包围体相交，如果光线确实与包围体相交，它就将沿着树向下遍历直到碰到一个实际对象，或者是什么也没有碰到。两条光线a和b如图所示。光线a与根包围体相交，然后将它和A+B做测试，结果是“击中”（即它确实与这个包围体相交）。进而让它分别与A和B做测试，结果是两个都不与它相交。它在树的这个层次上没有其他相交，因此这条光线被丢弃。注意，虽然该光线与许多包围体相交，但是相比于对场景中所有对象做相交测试，该方法有相当大的潜在节约。当然，一个不与光线相交的包围体可能包含数以千计的对象——光线就不需要与它们做比较测试。光线b不与A+B相交但与C相交，但它不与C内部的体素相交。它确实与D+E相交，而且最后与E内的体素相交。

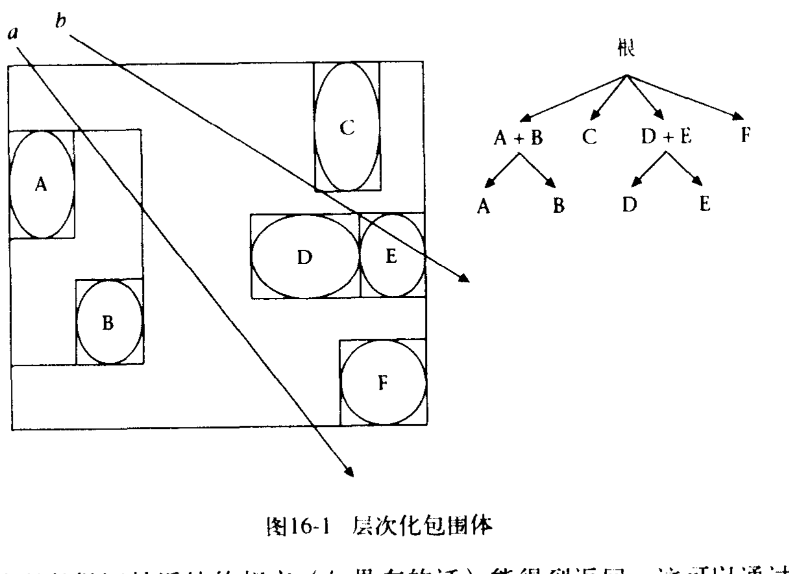


图16-1　层次化包围体

345 必须谨慎处理以保证最近处的相交（如果有的话）能得到返回。这可以通过对包围体预先排序来达到，所以光线是从原点到终端（在场景包围体内部）按照长度顺序向前处理的。这可以通过使用 BSP 数据结构实现，这里包围体的面用于优先权列表或其他类似数据结构，这些数据结构能充分利用包围体的面是轴向对齐这样一个事实——例如KD树（参见 Sanlet，1990）。

包围体的选择

遍历算法的代价取决于两个因素。第一个因素是光线和包围体之间相交测试的代价。与对象包围体的测试应该比与对象本身的相交测试要省时得多，否则包围体的优势就不复存在了。第二个因素是执行与包围体相交计算的总次数，这依赖于包围体与表面的紧凑程度以及包围体的层次结构。通常对于越紧凑的包围体，所需要的光线-包围体相交测试越费时，这是一种折衷情形。

一些简单的几何体形状被作为包围体，包括球体、正方体、椭球体、圆柱体和圆锥体。Kay和Kajiya（1986）提出了一种包围片方法。在这个方法中，一个片由两个平行的无限平面所定义。为了定义一个完全封闭的包围体，至少需要三个包围片。光线和片之间相交的t值由下式给出：

$$\frac{d_i-(n\cdot p_0)}{n\cdot v}, i=1,2 \tag{16-1}$$

这里n是片平面的法向，d_1和d_2是片平面和坐标原点之间的距离，p_0是光线源点，v是光线的方向（见式（8-8））。因为片的法向是已知且确定的，$n\cdot p_0$和$n\cdot v$只需要被计算一次。因此，相交降低为一次减法和一次除法。

层次化包围体方法能够在相交计算数目上获得一个大的缩减，但是在计算中每个对象（即使是只有它的包围体）都需要被光线考虑到。一个比较好的方法应该是这样的一种策略，即只有那些明显在光线路径上的对象才被测试，所有其他的都被忽略掉。我们将通过空间细分的方法来做到这一点。

16.4 一致空间细分

一致空间细分方法的思想是用轴向对齐的正方体来包围场景，并对空间进行一致地分割，所得到的每个小的立方体单元（有时称为“体素”）存储与之相交的对象的标识符。 346

这个思想是由Fujimoto 等（1986）、Amanatides（1987）以及 Cleary和Wyvill（1988）提出来的，它显著地减少了光线-对象相交的数目，因为光线经过单元的路径被列举出来，只有那些位于路径上的对象需要考虑相交。一旦发现与对象的第一个有效相交，光线路径就立即终止。

算法需要列举出穿过光线的那些单元。这是通过使用3DDA算法（数字微分分析器）的一个改换算法来获得的，这将在下一章中讨论。这个算法（见图16-2）说明光线是如何被跟踪的（以二维为例），这里光线所穿过的所有单元都必须考虑到。

图16-3说明有许多陷阱必须避免。首先，注意光线r将穿过两个单元，两者都包含对象A。显然对A测试两次是没有意义的。因此每条光线给定一个惟一的标识符，每个对象保存一个光线标识符列表，该列表保存的是已经做过测试的那些光线（如果有交点的话，还要将相应的交点记录下来）。第二，B和C是在同一个单元中相遇的。也许正好B先于C测试，并发现了相交。此时C会被错过，除非采用规则找出在当前单元中的所有相交点，并在当前单元里面对所有对象排序这些交点。光线s说明D和B将会在相同的单元中遇到。然而，B的相交点不发生在那个单元内部，因此在考虑那个单元里面的相交时就不需要将它包括在内。 347

一致细分模式有一个有用的特性，即通过空间细分计算光线路径是很省时的，正如我们

将在下一章中所看到的。主要的缺点是细分没有考虑到对象的分布。如果绝大部分对象都聚集在空间的一个角落，那么这种细分将是低效率的，因为在这个密集分布的区域中每个单元都会包含很多数量的对象，因此光线-对象相交计算方面的节约实质上会减少。

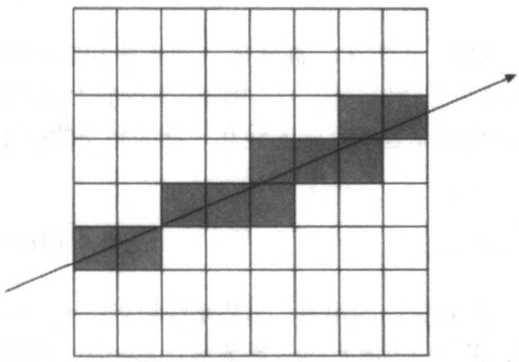


图16-2　通过空间细分跟踪一条光线（二维例子）

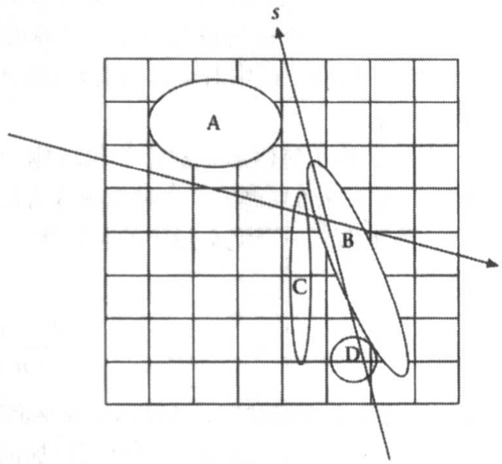


图16-3　只有当前单元中的相交才能终止光线

16.5　非一致空间细分

八叉树

一致细分方法对于对象在封闭正方体中均匀分布的场景来说，能取得比基本方法要快得多的效果。然而，如果对象都集中在场景的一个很小区域里面，那么在遇到第一个非空的单元之前，大量的空单元需要遍历。这倒不总是一个严重的问题，因为3DDA允许对单元进行非常快速的遍历。主要的问题在于对象都集中在相对很少数量的单元里，这些单元最有可能包括大量对象。因此，遇到这些单元的光线将需要大量的相交计算。更好的方法是使用自适应层次化细分以便空间根据对象的分布进行细分。在层次化细分中，如果一个单元包含超过预先设定的对象数量，单元将进一步细分。这允许我们快速跳过空的区域，同时也最小化了
348 相交计算的数目。除此之外，最大层次的细分也得到了维护，这样单元将不会被细分得太深。

典型使用的数据结构是八叉树。Glassner（1995）首先提出了在光线跟踪中使用八叉树。八叉树是一个层次化细分模式，允许空间依照对象的分布自适应地细分。在八叉树方法中，单元从每个轴的中间分割成2 × 2 × 2的八等份。八叉树的二维情形是*四叉树*，如图16-4所示。与一致空间细分模式相比，八叉树的遍历代价更高。为了要遍历内部单元的所有孩子，也可以使用一致细分方法用到过的渐增算法。然而，渐增方式在当有大量渐增步骤的时候更有效。但不幸的是，八叉树不是这种情况。另外一种方法是使用如下算法递归地将光线路径细分成一段段的。

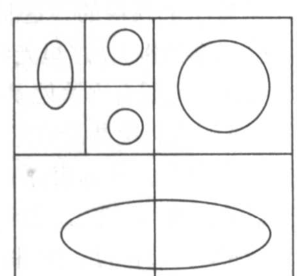


图16-4　四叉树——在这个例子中没有一个单元允许和两个或两个以上对象相交

为了要构造八叉树，世界被封闭在与轴对齐的正方体中。如果与正方体相交的元素数量少于一个最大的数目，那么停

止这个空间细分的过程。反之，在每一个轴向上将它一分为二，这样形成八个等份，再对每个等份递归地应用相同的原则。

这种方法的优势是明显的——细分是适用于场景中对象分布的。与一致细分相比较，其缺点是增加了光线穿过分割的代价。直接的执行过程如下：

- 光线与正方体相交，在内部轻微地移动它。
- 遍历八叉树看它位于哪个节点中。
- 与那个节点表示的立方体的面相交，找出它是从哪个面离开对应单元的，以便找到下一个单元。

BSP树

另一个非一致空间细分模式是使用BSP树。在一个轴向对齐的BSP树中，单元用轴向对齐的平面分割成两个孩子。每当细分深度增加的时候，依次选择三个主轴做上述分割。（这种树是KD树的三维实例，我们将会在第22章中再一次遇到它。）分割平面的位置可以以多种方式选择。举例来说，我们可以总是从它的中心细分一个单元。这产生八叉树数据结构的一个二叉树表示。缺点是它会造成冗余的细分，一个单元会被分割成两个单元：其中一个单元不包含任何对象，而另一个单元包含最初的所有对象。 349

比较好的做法是将对象平分到两个集合中。因此包围空间被分割到一个BSP树中，这里节点是与轴向对齐的平面。这个方法是由Kaplan（1985）和Jansen（1986）提出来的，具有下列特性：

- 如果分割平面位于主轴的中点，那么所得到的将是与八叉树一样的细分结果。
- 可以通过选择平面来获得一个自适应细分，比如总是将对象平分到两个集合中。

当使用BSP树时，光线遍历存在一个有效的迭代算法：

```
procedure BSPIntersect(ray,node) {
   if(ray interval empty or node == nil)  return;

   if(node is leaf then intersect ray with each associated object)
   else{
      near = ray clipped to near side of node plane;
      BSPIntersect(near, node->near);
      if(no intersection) {
         far = ray clipped to far side of node plane;
         BSPIntersect(far,node->far)
      }
   }
}
```

16.6 光线相关性方法

无论是将空间分割成没有重叠的单元或是包围体层次结构，都需要我们遍历产生的数据结构以便求出光线的路径。虽然这能带来非常显著的速度提升，但是也包含了遍历的代价，这个代价我们应当将它降为最小。降低这种代价的一个可能方法是考虑光线的相关性（Arvo and Kirk，1987；Ohta and Maekawa，1987）。利用光线相关性的意思是，对于给定一条光线来说，与它相交的对象的相邻对象很可能也与它相交。在这种情形中，我们可以缓存那些经过光线测试的对象，并在相邻光线处理中使用。这是测试阴影的有效方法。另一种方法是，

我们可以将一组光线用光线空间中的一个体包围住，枚举这个包围体的候选对象，同时枚举
350 正在与光线空间中光线做测试的那些对象。Arvo和Kirk（1987）提出了一个光线分类模式来分析这种光线相关性。他们执行在光线空间中的细分，而不是执行在对象空间中的细分。

我们将光线相关性思想与阴影感知器测试一同分析，然后再讨论Arvo和Kirk方法。

光缓冲区——高效的阴影测试

光缓冲区方法是由Haines和 Greenberg（1986）提出来的，这是一种能在发射阴影感知器光线时减少工作量的方法。该方法可以归纳如下：

- 构造一个虚拟正方体包围每一个光源位置。
- 考虑一个特殊的正方体——将环境投影到该正方体的面上，在每个面上使用矩形细分。
- 当阴影感知器击中一个正方体的面，检查对应的盖瓦。在这块盖瓦中的对象集合是那些有可能投射阴影到发出光线的对象。保持这个集合按深度排序。

光线分类

光线分类方法是与空间细分方法完全不同的一种方法，因为它是对光线空间进行细分而不是对对象空间进行细分（Arvo and Kirk，1987）。注意，一条光线可以表示为五维空间中的一个点：(x, y, z)是原点，(u, v)是方向。概括地说，这个方法用此来求出可能与一组“相似”光线相交的候选集合——光线在五维空间的某个相对较小的超立方体中。

该算法需要5个步骤：

（1）求出R^5的一个子集E，包含一个5D点，这个5D点等价于场景中每条可能的光线。E一定是一个有界集合。一条光线可以表示为5D点——方向通过选择场景中一个轴向对齐的立方体确定。每个面+X、－X、+Y、－Y、+Z、－Z由坐标系统表示、U和V在范围－1到+1之间。因此用UV表示的光线击中面的位置给出了额外两个自由度。因此一条光线可以表示为（x，u，z，U，V）和“主轴”（即（U，V）点所指的面）。所有的这种点（x，u，z，U，V）构成了有界子集合E。

（2）设E_1、E_2、…、E_n是E的分割（即每两个之间没有重叠且它们的并等于E）。E的分割是一个5D空间中类似八叉树的表示，即在5D空间中的超立方体可以沿着它的每个轴被一分为二，形成32个分割。每个这种分割可以类似地根据某些规则进行。六个主轴中的每一个都会有这样的32叉树。

351 （3）对于每个E_i有一个C_i，C_i表示所有至少与集合E_i中一条光线相交的对象集合。5D中的超立方体对应于3D中一个光束。这个光束所相交的对象集合是这个超立方体的候选集合。注意，当我们沿着32叉树前进的时候，与每个节点相交并关联的对象将会越来越少。

（4）对任何光线，求包含它的特定的E_i（从（2）知它是惟一的）。因为一条光线等价于一个5D点，求包含它的32叉树的节点可用一个简单的树遍历算法。

（5）给定光线和对象的一个集合C，求出相交。

全部的算法概要如下：

（1）设定初值。

```
Create the roots of the six 32-trees for each of the axes;
each 32-tree root, corresponding to the 5D hypercube, inherits
the entire set of objects.
```

（2）光线分类。

```
p = 5-tuple for the ray;
axis = dominant axis;
H = leaf hypercube of the 32-tree containing p;
```

（3）创建候选列表。

```
if C(H) is "inherited" then C(H) = C(H) ^ Beam(H);
while (C(H) > max_H and size(H) > min_H) {
   split H along 5 axes creating 32 children;
   let all new children "inherit" C(H);
   H = child which contains p;
   C(H) = C(H) ^ Beam(H);
};
```

（4）候选处理。

```
for each candidate in C(H) (considered in ascending order of
minimum extent) {
   check ray with bounding volume;
   if(intersects bounding volume) {
      intersect ray with object;
      if(intersects object){
         return this intersection;
      }
   }
}
```

最初我们建立六个32叉树，代表光线空间中的整个5D包围体。每棵树与整个对象集合关联。那么对任何光线求出它的主轴以及32叉树中包含它的单元（H）。如果与H关联的对象集合是“继承”的，那么具体地计算3D光束与这组对象的相交，以便收缩该集合。这种只在需要的时候计算集合的方法被Arvo和Kirk称为集合的“懒惰计算”。现在继续拆分和创建子节点，直到所创建的节点太小，或是该节点所对应的对象数目足够少。现在重新设包含p的节点是H，通过它相应的光束与集合中对象做相交测试，求出关联于H的对象集合。 352

候选处理利用这样的事实，即我们知道这条光线的主轴。因此对象根据在主轴方向上的最小尺寸排序。一旦发现相交点，所有在该主轴方向上最小尺寸超过相交点的对象都无需考虑，如图16-5所示。

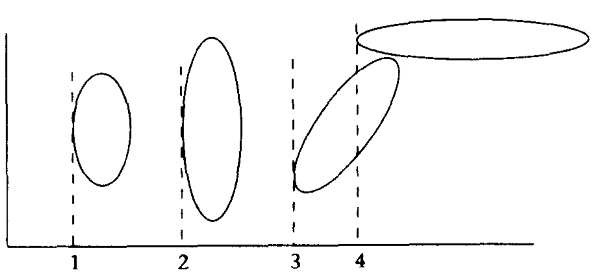


图16-5 在主轴上最小尺寸距离相交点很远的对象无需考虑

16.7 小结

本章简要地介绍了降低光线-对象相交数量的一些方法。正是这些方法使得光线跟踪在标准个人计算机或工作站上的实施变成可能。最近的一些进展表明，在个人计算机上实现实时

的光线跟踪已经不再是很遥远的事情了，有关这方面的情况可以参见Wald，Slusalleck and Benthin（2001）。

353 有很多重要细节并未在这一章中讨论，它们在计算机图形学中同样有广泛和重要的意义。第一是该如何有效地进行光线与包围盒的求交计算，第二是该如何在规则的空间细分中跟踪一条光线。这些将在下一章中讨论。

第17章 直线裁剪和渲染

17.1 引言

在上一章中我们用了各种各样的技术来加速光线跟踪。基本上来讲，这些方法主要是在各种不同的细分空间中跟踪光线，无论是一致细分或自适应细分空间。这里有两个问题：第一是如何高效地确定一条光线与另外的一个对象相交——尤其是细分空间的一个单元或一个包围体。在三维空间中这种体是立方体，在二维空间中它是矩形。在这一章中我们将把重点放在光线和矩形的相交确定上，其在更高维空间中的扩充是很简单的。这称为“直线裁剪”。在第10章中我们考虑过多边形的裁剪。

第二个问题是在细分空间中高效地计算光线的路径。假设有一个一致细分，要确定光线相交的单元顺序。目前这可以通过反复使用直线裁剪来解决：在整个场景中通过包围盒裁剪光线，并确定包含光线第一端点的开始单元，然后测试每个相邻单元和光线的相交并继续这个过程直到到达光线另一个端点。然而，这是非常低效率的——尤其是它没有考虑关联性—— 354
即利用一个单元的解来有效地求出光线将要进入的是邻近哪个单元。

虽然我们给出这个讨论的动机是出于快速光线跟踪的需求，但是还有另外一个非常强的动机是来自于二维计算机图形这个层次。在第5章中，我们将来自较高层次（世界空间）表示的对象映射到显示窗口。当然，这个映射过程不能保证对象被截取或裁剪——避免显示那些超出显示窗范围的对象部分。我们在第5章中使用多边形的处理，但是必须找到将直线裁剪到显示窗口的高效算法。

最后我们在第12章中看到了该如何在显示器上渲染多边形，也看到了如何对直线光栅化。如果一条直线是垂直的、水平的或者是有一个±1的斜度，那么问题是容易解决的。但是对于一般情况这不是一件容易的事情。求出一组尽可能接近给定线段的像素，这是二维计算机图形中的一个基本问题。对这个方法在高维上的推广也给了我们在一致细分三维空间中跟踪光线的方法。

在这一章中我们首先考虑用来裁剪直线的一些算法，然后考虑渲染直线的问题——即求出一组“最接近”给定线段的像素。

17.2 裁剪线段

直线方程及其相交

在这一小节中我们考虑直线和线段的数学表示。有三种类型方程可以用于直线表示：

直线的隐式方程（I）。其形式为：

$$Ax+By=C \tag{17-1}$$

其中A、B和C为常数。

隐式形式方程的一个很大好处是它能帮助我们确定任意点 $p=(x, y)$和直线之间的关系。

假定直线将空间分割成三个不相交部分。如果我们将直线方程表示成 $L(x, y)= Ax + By - C$，那么有三个集合：

- 位于直线上的所有点集合，这里$L(x, y)=0$；
- 位于直线一侧的所有点集合，我们称为正半空间，这里$L(x, y)>0$；
- 位于直线另一侧的所有点集合，我们称为负半空间，这里$L(x, y)<0$。

355 举例来说，假设有两点(x_1, y_1)和(x_2, y_2)，我们希望检查连接这两个点的线段是否与方程$Ax+By=C$所定义的直线相交。如果$L(x_1, y_1)$和$L(x_2, y_2)$的符号相反，则线段一定与直线相交。我们在第8章中利用到了这个方法。

直线的显式方程(E)。这是我们所熟悉的形式：

$$y=a+bx \tag{17-2}$$

这里b是直线的"斜率"，也就是说，x的一个单位变化所引起的y的改变量。值a是当$x=0$时在Y轴上的截取量。将形式I转换成形式E和将形式E转换成形式I都是非常容易的。换句话说，求出用A、B和C表示a、b的公式，以及用a、b表示A、B和C的公式很容易。

通过两点的直线(2P)。假设(x_1, y_1)和(x_2, y_2)是两个特定点，那么通过这两个点的直线方程是：

$$\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1} \tag{17-3}$$

设该直线不是完全水平或垂直的。

这也可以表示成隐式或显式的形式。

直线的参数化方程(P)。考虑用下式所描述的点：

$$p(t)=(\alpha_1+t\beta_1, \alpha_2+t\beta_2) \tag{17-4}$$

如果把t当作一个参数，那么对任何一个t值，我们有一个特殊点$p(t)$。举例来说，假设$p(t)=(1+2t, 3+4t)$，那么$p(1)=(3, 7)$，$p(2)=(5, 11)$，$p(3)=(7, 15)$。

很容易证明当t连续变化时$p(t)$点的轨迹是一条直线。事实上应该说明$p(t)$的公式满足式（17-1）和式（17-2）。

线段（LS）。上面的直线方程是描述无穷直线的。在计算机图形学中我们关心的是有限线段，即位于两点$p_1=(x_1, y_1)$和$p_2=(x_2, y_2)$之间的连续直线子集。使用参数化形式非常容易准确地定义一个线段上的点集合。考虑：

356

$$p(t)=(x_1+t(x_2-x_1), y_1+t(y_2-y_1)), t\in[0,1] \tag{17-5}$$

换句话说，我们考虑如前所定义的$p(t)$，但是这次将限制t在范围0到1之间（包含端点）。

显然$p(0)=(x_1, y_1)$和$p(1)=(x_2, y_2)$。对于在0和1之间的任意t，点$p(t)$在这两个点之间的线段上。举例来说，$p(0.5)$正好位于p_1和p_2的中央。

直线相交。直线和线段的不同形式表示都可以直接解决两直线之间的相交求解问题。假设我们有直线的隐式表示如下：

$$L(x, y)=Ax+By-C=0 \tag{17-6}$$

线段的表示如式（17-5）。

我们希望求出这两条直线之间的相交点（如果存在的话）。可以使用（1）中的讨论检查线段与直线是否相交。假设我们知道确实相交，那么到底相交发生在哪一点上呢？

在两直线相交点上，式（17-5）和式（17-6）都是满足的。换句话说，点$p(t)$一定是在直线$L(x, y)=0$上的，这意味着存在一个$t \in [0, 1]$，满足$L(x(t), y(t))=0$。经过整理有：

$$A(x_1 + t\,dx) + B(y_1 + t\,dy) = C \tag{17-7}$$

这里$dx=x_2-x_1$和$dy=y_2-y_1$。

如果我们对t求解这个方程，有：

$$t = \frac{C - Ax_1 - By_1}{A\,dx + B\,dy} \tag{17-8}$$

如果用这个结果替换$p(t)$中的t，我们得到所要求的相交点。图17-1说明了这样的思想。

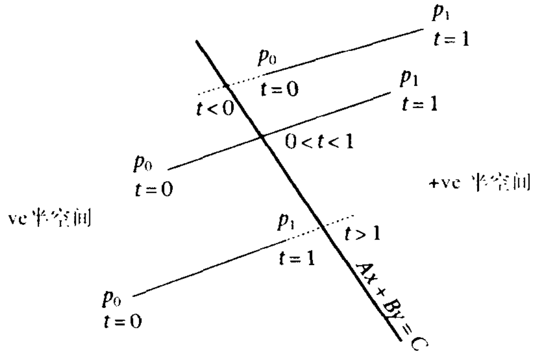


图17-1 参数化隐式直线方程和相交

二维裁剪区域

在这一小节中我们处理在二维矩形区域中的线段裁剪，这个矩形区域叫作“窗口”或“裁剪区域”。该区域由下式定义： 357

$$\begin{aligned} Xmin \leqslant x \leqslant Xmax \\ Ymin \leqslant y \leqslant Ymax \end{aligned} \tag{17-9}$$

特别地，有四条边界直线：

$$\begin{aligned} x&=Xmin \\ y&=Ymin \\ x&=Xmax \\ y&=Ymax \end{aligned} \tag{17-10}$$

每个边界的可见的一边是由相应的半空间确定的：

（1）$x > Xmin$是边界$x=Xmin$的可见的一边（这是直线$x=Xmin$的正半空间）。

（2）$x < Xmax$是边界$x=Xmax$的可见的一边（这是直线$x=Xmax$的负半空间）。

（3）$y > Ymin$是边界$y=Ymin$的可见的一边（这是直线$y=Ymin$的正半空间）。

（4）$y < Ymax$是边界$y=Ymax$的可见的一边（这是直线$y=Ymax$的负半空间）。

可见半空间的交集确定了裁剪区域。注意边界本身是在裁剪区域之内的。

Cohen-Sutherland 算法

Cohen-Sutherland直线裁剪算法（Newman and Sproull, 1979）是基于图17-2中所示的思想。在a中，直线完全位于裁剪区域之内，所以裁剪直线与原先直线一样。在b中直线完全在裁剪区域的外面，因为它的两个端点都位于左边界的左边。在c中直线部分在裁剪区域内，裁剪所得的结果如d中所示。

对于任何裁剪算法来说，其主要工作是计算所要裁剪的对象与裁剪区域之间的交。如果裁剪区域是矩形，则这些相交计算是简单的，因为它此时需要进行的是计算任意直线与一条水平和垂直边界的相交。

假如将要裁剪的线段是从（x_1, y_1）到（x_2, y_2），那么直线的方程是：

358

$$\frac{y-y_1}{dy}=\frac{x-x_1}{dx} \tag{17-11}$$

这里$dx=x_2-x_1$和$dy=y_2-y_1$。

这条直线与水平边界$y=Y$及垂直边界$x=X$的交点是：

$$\begin{gathered}\left(x_1+\left(\frac{dx}{dy}\right)(Y-y_1), Y\right)\\ \left(X, y_1+\left(\frac{dy}{dx}\right)(X-x_1)\right)\end{gathered} \tag{17-12}$$

然而，很明显使用式（17-12）计算直线与所有四个边界的相交是低效的——因为众所周知直线最多只能与裁剪边界中的两边界相交。Cohen 和 Sutherland 给出了一个算法，该算法试图减少所需要执行的相交计算次数，通过执行一个简单的测试来排除某些情况——例如当直线显然位于裁剪区域的外部或内部的时候。

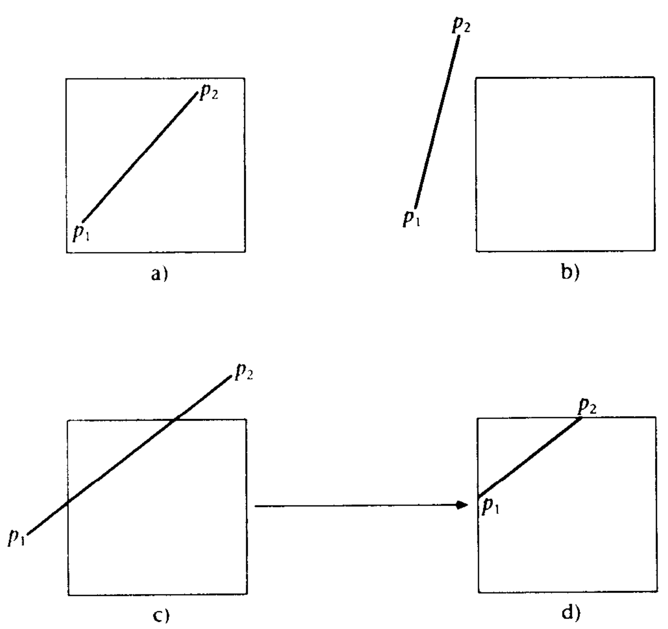


图17-2 裁剪直线

考虑图17-3，这是一个裁剪分类表。图中心处的被包围单元表示裁剪区域，那些周围的区域代表裁剪空间外部的象限区域。直线的两端点p_1和p_2可以关于这个表进行分类。设*Outside*是一个返回给定分类的函数。举例来说，对于图17-2b中所示的直线，p_1是低端的点：

$$\begin{aligned} &Outside\ (p_1)=\{Left\} \\ &Outside\ (p_2)=\{Left,\ Top\} \end{aligned} \qquad (17\text{-}13)$$ 359

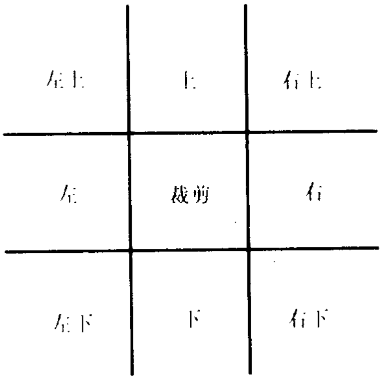


图17-3 裁剪分类表

我们可以这样看函数*Outside*，它返回点位于其外面的边界的集合。因此，如果两个点所对应的这种集合的交集非空，那么直线一定完全位于裁剪区域的外面——因为它的两个端点位于裁剪区域的左边、右边、下面或上面。对于图17-2b中直线这种情形，

$$Outside(p_1) \cap Outside(p_2) \neq \varnothing \qquad (17\text{-}14)$$

所以直线完全落在裁剪区域的外面。同样地，考虑图17-2a中的直线情形。它完全落在裁剪区域之内，所以它的两端点的位于其外的边界集合都应该是空集合。在这个例子中，

$$Outside(p_1) \cup Outside(p_2) = \varnothing \qquad (17\text{-}15)$$

因为两个端点都是在区域内部。

最后考虑一下图17-2c中的直线情形：

$$\begin{aligned} &Outside\ (p_1)=\{Left\} \\ &Outside\ (p_2)=\{Top\} \end{aligned} \qquad (17\text{-}16)$$

它既不满足式（17-14），也不满足式（17-15），所以就不能轻易地把它当作位于裁剪区域外部而排除，或者当作位于裁剪区域内部而接受。在这种情况下两个*Outside*集合的并包含直线的端点位于外侧的边界集合。选择其中任何一个边界，使之与直线相交并丢弃边界外面的 360
直线部分。现在对剩余线段重新执行这个过程。

整个裁剪过程因此依赖于两个函数——求直线与水平或垂直边界的交，以及求出点位于其外侧的边界集合。相交函数可以从式（17-12）构造，函数*Outside*非常简单，因为它只有点坐标与边界的比较。举例来说，假设坐标是（x，y），边界的方程如式（17-12）。

那么函数*Outside*会传送一个值outside，它的构造如下：

```
outside = Ø;
if x < Xmin then outside = outside ∪ {Left}
else
if x > Xmax then outside = outside ∪ {Right};
```

```
if y < Ymin then outside = outside ∪ {Bottom}
else
if y > Ymax then outside = outside ∪ {Top};
```

需要注意的是裁剪区域边界本身是在裁剪区域之内的。

实现Cohen-Sutherland 线段裁剪

对连接点p_1和p_2的直线裁剪:

- 设$o_1=Outside\ (p_1)$ 和$o_2=Outside\ (p_2)$。
- 如果$o_1 \cap o_2 \neq \varnothing$，那么将直线当作完全位于裁剪区域的外部而排除之。
- 如果$o_1 \cup o_2 = \varnothing$，将直线当作完全位于裁剪区域之内而接受之。
- 如果$o_1 \cup o_2 \neq \varnothing$，那么在$o_1$或$o_2$中选择其中一个边界并求直线与该边界的相交。称相交点为p_i。
- 如果被选择的边界来自于o_1，那么裁剪直线段p_i到p_2，否则裁剪直线段p_1到p_i。

361 在实现中，图17-3中区域通常使用四位编码表示（如图17-4）。现在求交和求并可以通过位或和位与操作来表示。（Dorr，1990）提供了算法的一个很好描述:

1001	1000	1010
0001	0000	0010
0101	0100	0110

图17-4 直线段分类的四位编码

```
c0 = region_code(p0);
c1 = region_code(p1);
while (c0 OR c1 ≠ 0) {
  if(c0 AND c1 ≠ 0) then reject line_segment;
  i = if c0 ≠ 0 then 0 else 1;
  j = position of most significant bit in ci ;
  pi = intersection point of line p0 p1 with boundary j;
  ci = region_code(pi)
}
```

参数化直线裁剪

上面给出的直线裁剪算法从知名度和使用广泛性上讲是最为普遍的。但这并不是说它是惟一的一个算法。这一小节我们要讨论另外一种称为 Liang-Barsky 的直线裁剪算法（Liang and Barsky，1984）。

这是一个所谓的参数化直线裁剪算法，因为它是基于两点（x_1，y_1）到（x_2，y_2）的直线段的参数化形式方程给出的，如我们在上面所讨论过的:

$$\begin{aligned} x(t)&=x_1+t\mathrm{d}x \\ y(t)&=y_1+t\mathrm{d}y \\ 0 &\leqslant t\leqslant 1 \end{aligned} \tag{17-17}$$

假设裁剪区域的定义如式（17-1）。利用式（17-17），这些不等式可以被重新写成:

$$\begin{aligned} -t\mathrm{d}x &\leqslant x_1 - Xmin \\ t\mathrm{d}x &\leqslant Xmax - x_1 \\ -t\mathrm{d}y &\leqslant y_1 - Ymin \\ t\mathrm{d}y &\leqslant Ymax - y_1 \end{aligned} \tag{17-18}$$

将这四个不等式改写成如下形式：

$$tp_i \leqslant q_i \quad i=1, 2, 3, 4 \tag{17-19}$$

这里，$p_1=-dx$，$q_1=x_1-Xmin$，i是对四个边界的标记，分别表示了边界的左边、右边、底部和顶部。对任意i，比值

$$r_i=\frac{q_i}{p_i} \tag{17-20}$$ 362

是直线和那个对应的裁剪边界之间交点的t值。

直线和裁剪边界之间的关系可以被分类为“进入”或“离开”。考虑当$p_i<0$的情形。那么，由式（17-19）得：

$$t \geqslant \frac{q_i}{p_i} \tag{17-21}$$

t的这组值表示了直线位于边界内部的部分。因为q_i/p_i是在相交点处的t值，而且对应于边界内部的t值来说，相交点具有最小的t值，那么这一定是一条“进入”直线。

同样地，当$p_i>0$时，这将是一条“离开”直线，即直线从边界i的可见一侧穿越到不可见的一侧。

当$p_i=0$时，不等式（17-19）减少到$q_i \geqslant 0$，所以如果$q_i<0$直线可以被排除。

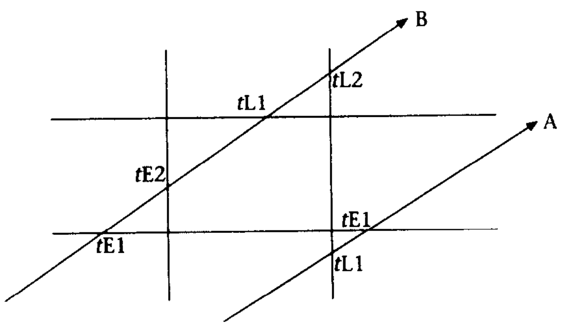


图17-5 参数化直线裁剪：$tE2$代替$tE1$，$tL1$代替$tL2$

算法背后的思想在图17-5中说明。任何（非垂直/水平）直线将会穿越所有的四个边界，对每个边界有一个对应的t值。每个t值可以被分类为“进入”（tE）或“离开”（tL）值。如果$tE>tL$那么直线一定位于裁剪区域的外面，如在直线A中。否则，所需要的直线是用最大的tE（$\geqslant 0$）和最小的tL（$\leqslant 1$）表示的，所以$tE \leqslant tL$。算法可以陈述如下。

参数化直线裁剪算法。设定初值$tE=0$和$tL=1$。

```
for each boundary (i = 1,2,3,4)
  if pi < 0 then
    if qi < pi then reject the line, since this is an E line yet
    ri > 1
    if qi ≥ 0 then do nothing, since ri would be negative, which
    cannot change tE
    if qi < 0 then if ri > tL then reject the line else tE =
    max(ri,tE)

  if pi > 0 then
    if qi < 0 then reject the line, since this is line is L, yet      363
    ri < 0
    if qi ≥ pi then do nothing, since ri ≥ 1, which cannot change tL
    if qi > 0 then if ri < tE then reject the line else tL =
    min(ri,tL).
  if pi = 0 then
    if qi < 0 then reject the line, since this contradicts the
    inequalities(EQ 17.18)
```

```
if the iteration has survived this far without a rejection of the
  line then the clipped line is given by the two parameter values
  tE and tL substituted into (EQ 17.17).
```

注意，如果在进入主循环之前就执行了琐细的排除测试（如在 Cohen-Sutherland算法中），那么情形如$p_i<0$，$q_i<p_i$和$p_i>0$，$q_i<0$就不会发生。同样地，如果首先执行接受测试的话，那“什么也不做”的情形就不会发生。

Nicholl-Lee-Nicholl直线裁剪

Cohen-Sutherland（CS）算法依赖于对直线端点的编码模式，来迅速排除那些端点位于特别裁剪边界外面的线段并迅速接受那些全部位于裁剪区域之内的线段。而对于其他线段与已知边界相交的情形，就要对截取直线继续重复这样的过程。Cyrus 和Beck（1978）以及如我们已经了解的Liang和Barsky（1984）介绍了另外一种方法，此时直线以参数化形式表示。对于LB这种情况，参数t的值对应于线段延长线与裁剪边界的交点，我们用该参数值求裁剪的直线。最初这些值是在0 和 1之间，对应于直线的端点，但通过与边界直线的相交渐进地“压缩”。Liang和Barsky 基于经验证据证明，对于有大量“随机”分布直线在显示空间的情形，以及对于裁剪区域具有各种不同尺寸的情形，这种参数化方法比 CS 方法更快。然而，这个观点是值得怀疑的（参见Slater and Barsky，1994）。

无论是CS 还是LB，都需要计算相交点，这些相交点可能不是裁剪直线的端点。Nicholl、Lee和 Nicholl（1987）（NLN）介绍了一种新方法，该方法只当新点明确地成为裁剪直线的一个端点的时候计算直线与裁剪区域边界的相交点。他们使用图17-3中的空间细分，但是基于直线的“第一个” 端点的位置增加了进一步的细分。一个特别情况如图17-6中所示，这里第一个端点（p_1）在单元 {L，T}内。第二个端点可能在任意另一个区域中，在区域中左边、顶端或底部的边界会与线段相交 （空的区域是不重要的排除情况）。对另外的可能情况我们有类似的分析。

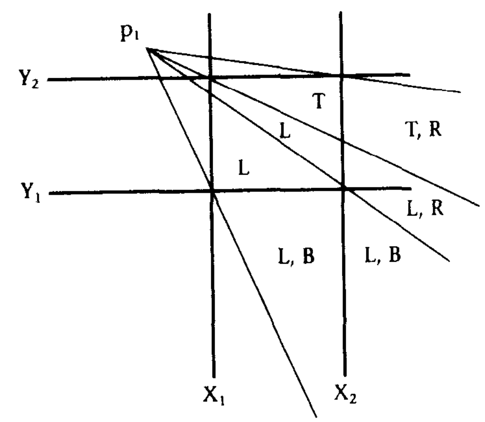


图17-6 NLN裁剪算法。粗线表示与直线相交的边界，交点依赖于第二个端点的位置

364 NLN说明他们的算法与其他知名的直线裁剪算法相比，需要的数学运算量是最小的。另一方面，虽然算法相当容易理解，但是由于大量的情况需要考虑，所以它的实现是相当冗长的。

这三个算法：CS、LB和NLN是计算机图形学中著名的直线裁剪算法。然而，还有很多它们的变种算法是鲜为人知的，另外在这方面还有其他的算法。对CS算法的改进主要有 Andreev（1989）、Duvanenko et al.（1990）、Shi et al.（1990）以及Blinn（1991），他们专注于通过最小化线段端点编码中的工作来加速算法。Liang和Barsky（1990）通过减少便于更多比较的分区数目的方法来改善最初的LB算法。Dorr（1990）通过证明LB算法可以完全用整数运算对算法做进一步的改进。Sobkow 等（1987）发展了另一个类似于 NLN算法的算法，之所以说它类似是实质上的类似（如果说不是在形式上的类似的话），这是因为他们枚举直线和裁剪区域之间的所有可能关系，只在需要作为部分输出的时候才计算相交点。

另外一个方法是Slater和Barsky（1994）提出来的，该方法在图17-3所示的细分空间中跟踪线段，很像Bresenham算法（见这一章中稍后部分）在像素的网格中跟踪线段。这个算法的优势在于它是“自适应的”，因为边界的处理顺序是根据它们沿着线段所发生的顺序，而不是如其他各个算法那样采用固定的顺序。

直线段与多边形裁剪区域和一般裁剪问题的复杂度已经得到了计算几何学文献的重视（即众所周知的直线-多边形分类问题，简称LPC）。举例来说，在Tilove（1981）、Skala（1989）和Rappaport（1991a）中都讨论过。 365

17.3 线段的光栅化

线段经过了裁剪，而且是可见的，现在就可以显示了。在这一小节中我们考虑该如何进行显示。假设有一个帧缓冲区以及一个设定单个像素的函数，我们需要计算对应于线段的像素位置。

我们立刻遇到了计算机图形学的一个基本问题——采样和走样。一条直线是一个连续、无限细的实体，然而我们只能在光栅网格不连续的位置上对它采样。像素不是无限小的，而是有一个有限的面积，从像素发出的光能所照亮的区域是发散的。因此在一个光栅显示器上渲染的直线是一个相当粗笨、不连续的锯齿形对象——根本不是一条直线！举例来说，如图17-7中所示，其描述了为表达一条线段有可能选择的大量像素。通常被显示的直线存在一种不正常，称之为走样现象——即在理想描述中不存在的方面，但是出现于对连续实体在离散网格上的采样过程中。然而，如果有足够大的显示和颜色分辨率，它就能造成人类视觉系统的错觉，直线看起来仍然是直线。在这一小节中，我们把重点集中在计算对应于一条直线的“最佳”像素位置上。

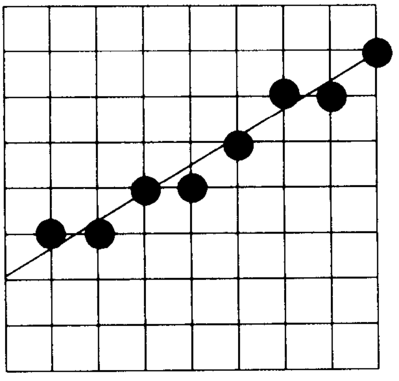


图17-7 直线段的理想光栅化

一个简单方法

假设直线是从（x_1，y_1）到（x_2，y_2），斜率dy/dx由式（17-11）给出。我们进一步假设直线的斜率大于0且小于1。从式（17-2）和式（17-3）能看出计算对应于直线段的像素的一种可能算法如下所示： 366

```
void line0(int x1, int y1, int x2, int y2, Color color)
{
   float y;
   int x;
   for(x = x1; x <= x2; ++x){
      y = (dy/dx)*(x-x1) + y1;
      setPixel(x,round(y),color);
   }
}
```

这个方法是朴实的。它忽略了在x和y之间线性关系的特性，即对x的每个单位增量，y按照给定斜率 dy/dx以固定常量递增。因此可以有一个更好的算法：

```
float line1(int x1, int y1, int x2, int y2, Color color)
{
   float y = y1 , m = dy/dx;
   setPixel(x1,y1,color);
   for(x=x1+1; x<=x2; ++x){
      y = y + m;
      setPixel(x,round(y),color);
   }
}
```

line1在很多方面都优于line0，但是即使是line1也包括浮点算术，包括除法和取整。事实上只使用具有加法和以2为乘数乘法的整数算术来构造画线算法是可能的。这个算法是由Bresenham（1965）提出来的，也可参见 Foley et al.（1990）。

Bresenham画线算法

在这一小节中，我们假设$x_1=y_1=0$，$dx>0$，$dy>0$，而且斜率大于0小于1($0<dy<dx$)。这些限制只是在解释算法概念时有用，它们很容易在实现中克服掉。

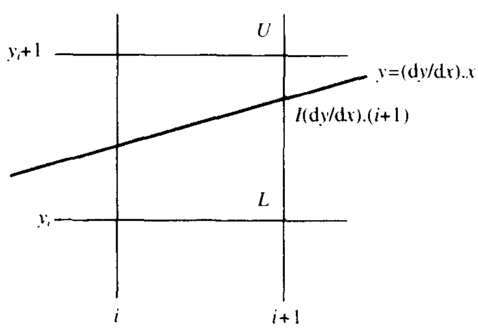


图17-8 在像素U和L之间选择

我们的任务是找出像素，它们在某种意义上最"靠近"直线。在图17-8中，像素位置位于垂直直线和水平直线的交点上，假设要显示的直线在所示的区域中。现在采用的准则是，对每个连续的x值，我们将选择与垂直直线具
367 有最小距离的像素。那么在像素$x=i$处，选择像素（i、y_i），问题是要确定直线下个像素点是选择 U（$i+1$，y_i+1）还是选择L（$i+1$、y_i）。

直线与 $x=i+1$相交于点 $\left(i+1,\left(\frac{dy}{dx}\right)(i+1)\right)$。我们称这个点为$I$（意为相交）。依照我们的标准，如果从$U$到$I$的距离小于从$I$到$L$的距离，那么应该选择$U$，否则选择 L。（如果距离是相等的，选择哪个都一样。）因此，

$$\text{如果}U-I<I-L\text{，选择}U\text{，否则选择}L \tag{17-22}$$

这等价于:

$$\text{如果 } y_i+1-\frac{dy}{dx}(i+1)<\frac{dy}{dx}(i+1)-y_i\text{ ，选择}U\text{、否则选择}L \tag{17-23}$$

因为根据我们的假设 $dx>0$，在两边乘 dx，不改变不等式的意义:

$$\text{如果}dx(y_i+1)-dy(i+1)<dy(i+1)-y_i dx\text{，选择}U\text{、否则选择}L \tag{17-24}$$

现在将所有项都移到不等式的左侧，有:

$$\text{如果}dx(2y_i+1)-2dy(i+1)<0\text{，选择}U\text{，否则选择}L \tag{17-25}$$

设

$$e_i=dx(2y_i+1)-2dy(i+1) \tag{17-26}$$

那么我们的条件变成:

$$\text{如果}e_i<0\text{，选择}U\text{，否则选择}L \quad (17\text{-}27)$$

对所有i用 $i+1$ 替换，在式（17-26）中：

$$e_{i+1}=\mathrm{d}x(2y_{i+1}+1)-2\mathrm{d}y(i+2) \quad (17\text{-}28)$$

用式（17-28）减去式（17-26），我们可以看到：

$$e_{i+1}=e_i+2\mathrm{d}x(y_{i+1}-y_i)-2\mathrm{d}y \quad (17\text{-}29)$$

现在考虑"选择U"和"选择L"的含义。"选择U"意味着$y_{i+1}=y_{i+1}$，"选择L"意味着$y_{i+1}=y_i$。因而我们可以将式（17-27）重写成：

$$\text{如果}\ (e_i<0)\{y_{i+1}=y_i+1\},\text{否则}\ \{y_{i+1}=y_i\} \quad (17\text{-}30)$$ 368

现在使用式（17-29）：

$$\begin{aligned}&\text{如果}\ (e_i<0)\{y_{i+1}=y_i+1;e_{i+1}=e_i+2(\mathrm{d}x-\mathrm{d}y)\}\\&\text{否则}\ \{y_{i+1}=y_i;e_{i+1}=e_i-2\mathrm{d}y\}\end{aligned} \quad (17\text{-}31)$$

该公式所说的是在每个连续的x坐标上检查变量e的符号。如果它是负的，那么我们选择上方的y坐标，同时将变量e增加2(dx - dy)；否则选择比较低的y坐标，同时将变量e减去2dy。

变量e是一个误差项。它是上方像素的y值与直线段和垂直直线在x值处交点的y值之间的差值，然后乘以 dx，将它变成一个整数值。

为了构造一个完全的算法，我们需要找到e的一个适当的初始值即e_0。将式（17-26）中i用0替换，得到

$$e_0=\mathrm{d}x-2\mathrm{d}y \quad (17\text{-}32)$$

根据我们先前的假设，有$y_0=0$。

现在将各个方面综合在一起，我们构造如下的程序。

Bresenham算法。直线始端在(0，0)，终端在(dx，dy)，dx>0，dy>0，且dy/dx<1。

```
void bresenhamLine(int x1, int y1, int x2, int y2, Color color)
{
   int dx = x2-x1, dy = y2-y1;

   int e = dx - 2*dy;
   int y = 0;
   setPixel(0,0,color);
   for(int x=1; x<= dx; ++x) {
      if(e < 0){
            y = y + 1;
            e = e + 2*(dx - dy);
      }
      else{
           e = e - 2*dy;
      }
      setPixel(x,y,color);
   }
}
```

实际上，我们当然不会使用乘法来计算乘2运算，而是使用位移来完成的。举例来说，在C语言中 `2*y`可以通过`y<<1` 计算。

算法是针对一个特殊情况开发的——它假设直线开始于原点，x是主轴(0< dy<dx)。如果直线不是从原点开始的，那么算法就需要作一点改变，通过设定y的初始值为y_1且x的初始值为 369

x_1。如果x不是主轴，那么就交换算法中x和y的角色。如果$dx<0$，那么用减量而不是用增量，而且对于直线是完全水平或垂直的情况还需要特别地处理。Nicholl和Nicholl的一篇文章（1990）给出了构造对应于几何变换的程序变换，并对 Bresenham 算法的各种不同情况当作例子处理。

我们已经介绍了线段光栅化的算法核心。暂时不考虑使走样复杂化的因素，也不对这个核心给予太多的改善。然而，十分重要的一点是，已经在研究致力于改善 Bresenham 算法的性能。基本思想不是仅仅查看下一步哪个像素有可能选取（U或L），而是考虑多步（比如说四步）的可能模式。通过这种方法可能很快地识别出哪一组连续四个像素应该被设定而无需做任何算术计算。这些工作以及类似的一些工作可以参见Rokne and Wyvill（1990），Rokne and Rao（1992），以及 Fung et al.（1992）。Bresenham 画线算法的类似技术也应用于其他基本体素，例如圆和椭圆（见 Pitteway，1967; Bresenham，1977）。

17.4 在一致细分空间中跟踪光线

Bresenham 算法对于二维图形是基础，但是我们由于光线跟踪中的问题而激发了对它的兴趣：在一致细分空间中高效地跟踪一条光线，其原因如我们在："一致空间细分"一节所讨论的。虽然求解一组最佳像素对应于一条直线的问题与求解光线在一致细分空间中遍历一组单元的问题是相似的，但我们仍不能简单地改编 Bresenham算法来解决光线遍历问题。更重要的一点是Bresenham算法的思想，正是这种思考问题的方法才能使得我们将它应用到略有不同的上下文中。

图17-7说明了求解对应于一条直线的像素的问题。将它与图17-9做比较。在光线跟踪情况（当然在3D空间中）下，我们需要枚举光线经过的单元。在二维例子中，对于光线A，遍历算法应该报告：(0，2)，(1，2)，(1，3)，(2，3)，(3，3)，(3，4)，(4，4)，(5，4)，(5，5)，(6，5)。对于光线 B，它应该报告：(1，0)，(2，0)，(2，1)，(3，1)，(4，1)，(5，2)，(6，2)。

370 注意在光线B中发生而没有在光线A中发生的事情。对于B这种情况，当每个单元坐标增加1（从（4，1）到（5，2））的时候有一个单元转换的过程，这涉及到遍历算法所容许的连通性。考虑 3D 情况，假设单元坐标标记为（u_1，u_2，u_3）。现在当光线移动到下个单元内，每个坐标要么保持不变，要么增1，要么减1。举例来说，下个单元的坐标可能是（$u_1+\Delta u_1$，$u_2+\Delta u_2$，$u_3+\Delta u_3$），每个$\Delta u_i=0$或者± 1。这里有27种可能的转换。去掉空转换，即每个$\Delta u_i=0$，我们可以得到从一个单元到下一个单元的26种可能的转换。这称为*26连通路径*。有关于此的另外一个思想是，下一个转换中光线可能穿越立方形单元的其中一个面（有6种可能性，这里只有惟一的u_i发生改变）；或者正好穿越其中的一个边（有12种可能性，这里正好有两个u_i发生改变）；或者正好经过其中一个顶点（有8种可能性，这里所有三个u_i均发生改变）。因此全部有26种可能的转换。一个遍历算法支持所有26种可能的转换则称之为*26连通算法*。当然，也有*18连通算法*（算法对于每个单元变换只报告一或两个坐标变化），以及*6连通算法*，这里在每个转换中只允许一个u_i改变。在光线跟踪的上下文中，很重要的一点是努力去最小化遍历所报告的单元集合，因为对于每个列举的单元，我们必须求光线与这个单元中所存储的一组对象标识符所标识的对象的交，而这是一个代价很高的操作。因此我们应该报告26连通路径。另一方面，我们总是用浮点数运算，存在一定程度上的不精确，所以不太可能有光线正好经

过单元的一个顶点的情况发生。在这一小节的余下部分中我们将只研究6连通算法（26连通算法的完全导出可以参见Slater，1992b）。

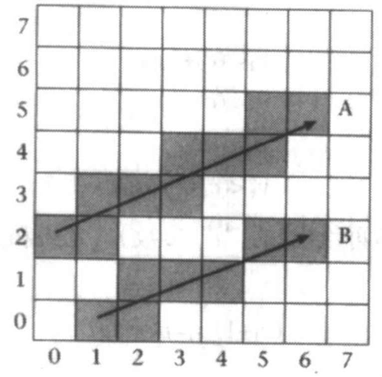


图17-9 光线遍历一致细分空间中的单元

假设每个单元的宽度(X轴)、深度(Y轴)和高度(Z轴)分别是s_1、s_2和s_3。设光线是$p=(p_1, p_2, p_3)$到$q=(q_1, q_2, q_3)$，有$du_j=q_j-p_j$，$du=(du_1, du_2, du_3)$。不失一般性，我们假设$du_j>0$，$p_j>0$。因此光线的方程是:

$$p(t)=p+t\,du \qquad t\in[0,1] \tag{17-33}$$

当t从0变化到1时，光线将与单元相交于连续的整数坐标

$c(0), c(1), \cdots, c(m)$，这里:

$c(i)=(c_1(i), c_2(i), c_3(i)) \quad i=0, \cdots, m$

且 (17-34) 371

$c(0)=(p_1/s_1, p_2/s_2, p_3/s_3)$

$c(m)=(q_1/s_1, q_2/s_2, q_3/s_3)$

这里除法也只取它们的整数（举例来说，用这种表示法，有21/5=4）。

图17-10展示了一个特别的单元，坐标为（c_1，c_2，c_3）。注意实际 3D 坐标是如何通过分别乘以宽度、深度和高度获得的。

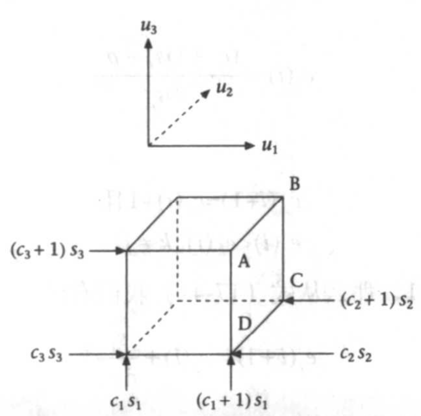


图17-10 坐标为（c_1，c_2，c_3）的单元

假如光线进入单元$c(i)$，考虑当$c_1(i+1)=c_1(i)+1$的情况，所以转换是沿着x方向发生的。这只有在光线与单元面ABCD相交的时候才有可能发生。假设相交点是(v_1, v_2, v_3)，那么这意味着：

$$v_2 < (c_2+1)\,s_2$$
$$v_3 < (c_3+1)\,s_3 \tag{17-35}$$
或
$$v_k < (c_k+1)\,s_k,\ k \neq 1$$

（我们有时为了书写方便不再给出c的“i”参数）。光线和方程为$u_1=(c_1+1)s_1$的单元面相交发生在下式成立的时候：

$$(c_1+1)\,s_1 = p_1 + t\mathrm{d}u_1 \tag{17-36}$$

因此：

372 $$v_k = p_k + ((c_1+1)s_1 - p_1)\frac{\mathrm{d}u_k}{\mathrm{d}u_1} \tag{17-37}$$
$$k = 1,2,3$$

使用式（17-37）和式（17-35），得：

$$c_1(i+1) = c_1(i)+1 \text{ 且}$$
$$p_k + ((c_1+1)s_1 - p_1)\frac{\mathrm{d}u_k}{\mathrm{d}u_1} < (c_k+1)s_k,\ k \neq 1 \tag{17-38}$$

对c_1的导出没有什么特殊的地方，对任何其他的c_j可以采用类似的讨论。我们得到：

$$c_j(i+1) = c_j(i)+1 \text{ 且}$$
$$p_k + ((c_j+1)s_j - p_j)\frac{\mathrm{d}u_k}{\mathrm{d}u_j} < (c_k+1)s_k,\ k \neq j \tag{17-39}$$

记住我们现在正在导出6连通算法，所以在每次迭代中有一个且只有一个c_j会增加。

整理式（17-39）得：

$$\frac{(c_j+1)s_j - p_j}{\mathrm{d}u_j} < \frac{(c_k+1)s_k - p_k}{\mathrm{d}u_k} \tag{17-40}$$

写成：

$$e_j(i) = \frac{(c_j+1)s_j - p_j}{\mathrm{d}u_j} \tag{17-41}$$

现在算法可以陈述为：

$$c_j(i+1) = c_j(i)+1 \text{ 且}$$
$$e_j(i) < e_k(i),\ k \neq j \tag{17-42}$$

假设情况是$c_j(i+1)=c_j(i)+1$，那么从式（17-41）我们有：

$$e_j(i+1) = e_j(i) + \frac{s_j}{\mathrm{d}u_j} \tag{17-43}$$

在式（17-41）中取$i=0$，使用式（17-34）有：

$$e_j(0)=\frac{((p_j/s_j)+1)s_j-p_j}{\mathrm{d}u_j}=\frac{s_j-(p_j\%s_j)}{\mathrm{d}u_j} \quad (17\text{-}44)$$ 373

这里"%"是"mod"或余数算子。我们对"error"项设定初值，该项与 Bresenham 算法中的"e"扮演的是相同的角色。从所有这些元素中我们能整合出下列算法:

```
void UniformRay(p, q, s1, s2, s3)
{
  /*initialization*/
  for(j=1,2,3){
    duj = qj - pj;
     cj = pj/sj;
    cqj = qj/sj;
      ej = (sj - (pj%sj)) / duj;
      rj = sj / duj;
  }
  Report(c);

  /*main loop*/
  while(c ≠ cq){
    for(j=1,2,3){
      if(for each k ≠ j ej < ek){
        cj = cj + 1;
        ej = ej + rj;
        break; /*out of for loop*/
      }
    }/*end for*/
    Report(c);
  }/*end while*/
}
```

Report意思是将对应单元坐标返回给调用程序（光线跟踪程序）并进行处理。这个算法类似于由Cleary 和 Wyvill（1988）针对所有s_j都相等的情形提出的算法。

报告所有在一致细分空间中被光线遍历的单元的算法有相当长的历史了，如我们所能看到的，它们的导出中使用了和 Bresenham 算法相似的推理。这些算法在Dippe and Swensen（1984）、Glassner（1984）、Kaplan（1985）、Fujimoto et al.（1986）、Kaplan（1987）、Amanatides（1987）以及Cleary and Wyvill（1988）都有讨论。关于在非一致细分空间光线跟踪问题的最新结果可参见 Havran and Bittner（2000）。

17.5 小结

在这一章中我们考虑了基本几何图元：直线。在本章前面部分中我们主要讨论了高效实现在矩形边界中裁剪一条直线的问题。这种灵感来自于光线跟踪中包围体或单元与光线的相交。我们给出了在二维空间中的方法，它本身就是很重要的。其在三维空间中的扩充是简单的，在第10章多边形裁剪中所提出的思想对此有帮助作用。Cohen-Sutherland算法以及Liang-Barsky算法容易被扩充到三维空间中，但NLN算法不能扩充到三维空间。

在本章后一部分中我们考虑在二维网格（一个像素化显示空间）中高效渲染直线的问题，
374 并导出了Bresenham算法。这一点的最初动机来自于在三维细分空间中光线路径的跟踪。而后返回到在一致细分空间中跟踪光线的问题，并说明了这种算法是如何通过类似于 Bresenham 算法的讨论导出的。

第四部分　实体、曲线和曲面

第18章　体素构造表示

18.1　引言

我们前面从针对球的光线投射和光线跟踪开始，然后转向专门处理多边形。这一章和下一章将介绍其他类型的基本体素——在这一章中我们主要介绍二次曲面，这本质上是对象的一个一般类，球面是它的一个特别的情况，它包括基于隐式二次方程的实体表示和组合。在下一章中我们考虑另外的一种形式，利用高阶参数化表示的边界表示——B样条曲线和曲面。

本章也要介绍体素构造表示（CSG），它提供了实体对象的表示方法和构造方法。这个思想是将基本体素对象用 3D 空间中点集来表示，然后通过并、交、差运算来构成较为复杂的对象。这些基本体素可以是半空间，也就是说，某平面一侧的所有点。在这一章中我们使用二次曲面作为基本体素，而半空间是二次曲面的一种特殊（退化）情况。 375

在这一章中，对CSG对象的渲染是借助光线投射方法完成的，关于光线投射在第5章中已经介绍过。我们可以和读者一起回想一下，这是第一层的“反向光线跟踪”，这里光线是从投影中心经过显示屏幕上的每个像素，跟踪到最近表面的相交点。在这个方法中进一步的反射和折射光线不再考虑。

集合操作对CSG而言是基础。然而，当用点集来表示实体时，简单地对这些实体应用并（∪）、交（∩）和差（－）算子会得到非实体结果。这由图18-1说明，这里立方体A和 B 相交得到对象 C，它只是一维对象。

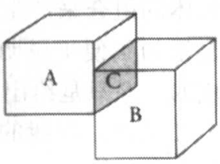


图18-1　A∩B=C

为了要克服这个问题，我们需要使用所谓的*正则集合表示*。为了解释这一点，我们介绍来自于拓扑学的一些简单思想。假设空间R^3和一个点$p\in R^3$，且$\varepsilon>0$，定义

$$B(p,\varepsilon)=\{q\in R^3 \mid \|q-p\|<\varepsilon\} \qquad (18\text{-}1)$$

这里$\|q-p\|$是从q到p的距离。$B(p,\varepsilon)$表示点的一个开球，p点位于球的中心。

S的一个*边界点*是任意点p，使得$B(p,\varepsilon)$包含S内的点，也包含对于任意ε来说$\sim S$(S的补集)内的点。S的*边界*可能是也可能不是S的子集。如果S不包含它的边界，则S是一个*开集合*。如果它确实包含它的边界，则S是一个*闭集合*。S的闭包定义如下:

$$closure(S)=S\cup boundary(S) \qquad (18\text{-}2)$$ 376

S的*内部*是S减去其全部边界点。

S的*正则化*是$interior(S)$的闭包。这由图18-2说明。

给定任何布尔集合运算，我们需要的是它的正则化结果。因此如果op是交（∩）、并（∪）或差（－）运算之一，那么正则化布尔算子定义如下:

$$A\ op^{*}B = closure(interior(A\ op\ B)) \tag{18-3}$$

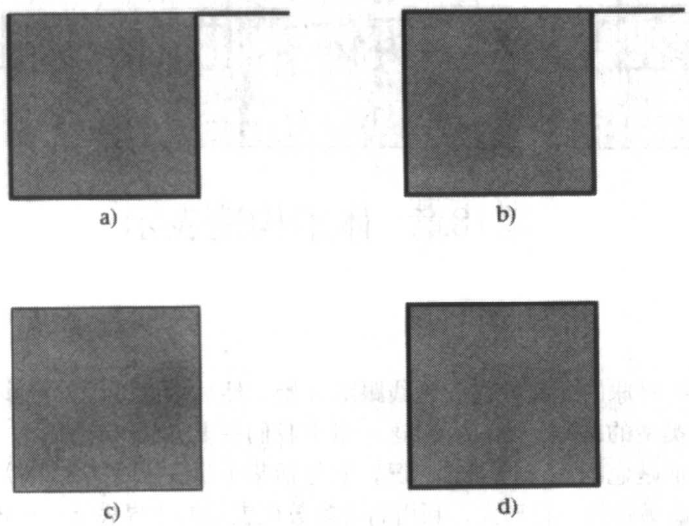


图18-2 正则集合操作：集合a封闭于集合b，它的内部是c，它的内部(正则化)闭包是d

18.2 二次曲面

这一小节我们将讨论 CSG 思想以及在这样的上下文中光线投射的使用。标准的参考文献是Roth（1982），Roth给出了如何通过基本体素例如方块体、圆柱体、球体、圆锥体和圆环体的组合来产生更复杂的形状。组合是通过集合算子并交差实现的。一个例子如图18-3，这里为了便于说明问题，用一个圆柱体穿透一个方块体形成一个圆洞，这是使用减算子完成的。这只是给出了一个单层运算。当然，表达式是可以任意复杂的，CSG公式因此形成一个二叉树。在树的叶节点上是基本体素，内部节点表示整个对象的子部件，整个对象由根节点表示。

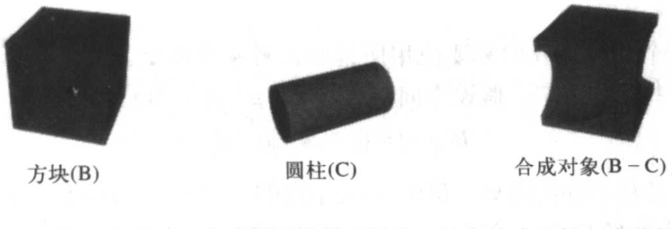


图18-3 CSG的一个例子：合成的对象是通过从一个方块中抽掉一个圆柱所形成

首先考虑交运算。经过两点光线的参数化方程已经在上一章中给出。在目前的上下文中，其中一个点是COP，而另一个点代表光线所穿过的像素。光线与方块的交本质上与先前一章中讨论的3D方盒裁剪中所使用的是相同的计算。这里要考虑的另外一种情况是关于二次曲面（举例来说能用来表示圆柱体、圆锥体和球体的曲面）。（可以参见Heckbert，1984。）一个二次曲面可以用二次方程表示：

$$s(x,\ y,\ z)=pQp^{T}=0 \tag{18-4}$$

这里p是行向量$(x,\ y,\ z,\ 1)$，Q是一个对称的4×4常数矩阵。

二次实体可以用不等式表示为：

$$s(x,\ y,\ z)=pQp^{\mathrm{T}}\leqslant 0 \qquad (18\text{-}5)$$ 377

因此一个二次曲面将空间分割为三个区域：在二次曲面表面上的点、在二次曲面内部的点以及位于二次曲面外部的点。

光线对二次曲面的投射是简单的。显然球体是一个特别情况，这已经在第5章中详细地讨论过。按照如下方式很容易一般化为对任何二次曲面的情形。光线方程可以表示为：

$$p(t)=c+td \qquad (18\text{-}6)$$

这里c是光线原点，d是光线的方向矢量。

将式（18-6）替换到式（18-4）中，得到二次方程的t参数表示。根据是否光线与二次曲面相交，根要么为实数要么为虚数（假设Q是满秩的）。换句话说，光线与非退化二次曲面的相交情况要么相交于其上的两点，要么根本不相交。

可以通过适当选择Q的表达式中的常数a到j，二次曲面能够用来表达多种有用的实体，Q的一般形式为：

$$Q=\begin{bmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{bmatrix} \qquad (18\text{-}7)$$

如果展开式（18-4），可以得到：

$$\begin{aligned} s(x,y,z) &= ax^2+ey^2+hz^2+2bxy+2cxz+2fyz+2dx+2gy \\ &\quad +2iz+j=0 \end{aligned} \qquad (18\text{-}8)$$

现在通过适当选择在Q中的常数，我们能获得如下的每一种实体。

平面

$$\begin{aligned} &Ax+By+Cz-D=0 \\ &a=e=h=b=c=f=0 \\ &2d=A,2g=B,2i=C,j=-D \end{aligned} \qquad (18\text{-}9)$$

球面

$$\begin{aligned} &x^2+y^2+z^2=r^2 \\ &a=e=h=1 \\ &b=c=f=d=g=i=0 \\ &j=-r^2 \end{aligned} \qquad (18\text{-}10)$$

圆柱面

$$\begin{aligned} &x^2+y^2-1=0 \\ &a=e=1 \\ &h=b=c=f=d=g=i=0 \\ &j=-1 \end{aligned} \qquad (18\text{-}11)$$ 378

圆锥面

$$
\begin{aligned}
&x^2+y^2-z^2=0\\
&a=e=1\\
&h=-1
\end{aligned}
\tag{18-12}
$$

所有的其他系数为0

抛物面

$$
\begin{aligned}
&x^2+y^2+z=0\\
&a=e=1\\
&2i=1
\end{aligned}
\tag{18-13}
$$

所有的其他系数为0

双曲面（双叶）

$$
\begin{aligned}
&x^2+y^2-z^2+1=0\\
&a=e=1\\
&h=-1\\
&j=1
\end{aligned}
\tag{18-14}
$$

所有的其他系数为0

双曲面体（单叶）

$$
\begin{aligned}
&x^2+y^2-z^2-1=0\\
&a=e=1\\
&h=-1\\
&j=-1
\end{aligned}
\tag{18-15}
$$

所有的其他系数为0

对于圆环面，虽然它不是一个二次曲面，也可以相对容易地在光线投射中使用，因为相交方程是四次的，可以得到其解析解。

圆环面

$$(x^2+y^2+z^2-(a^2+b^2))^2=4a^2(b^2-z^2) \tag{18-16}$$

这表示一个圆环面，横截面将是两个圆环，其与中心的距离是 $2a$，半径为 b。

Roth也使用过单位立方体作为一个基本体素。

对于二次曲面这种情况，在曲面上点 $p=(x, y, z)$处的法向特别容易求出。法向是如下矢量：

379

$$2pQ \tag{18-17}$$

最后，很容易证明如果属于二次曲面的所有点p经过转换矩阵M转换，那么所产生的表面也是二次曲面。

设$q=pM$是转换后的二次曲面上的点，对应于原二次曲面表面上p点。那么假设M是满秩的：

$$qM^{-1}=p \tag{18-18}$$

利用式（18-5）得：

$$qM^{-1}Q(qM^{-1})^T=0 \tag{18-19}$$

如果我们写成 $R=M^{-1}Q(M^{-1})^T$，那么就能够清楚地看出转换后所得到的点q仍然属于一个二次曲面。

18.3 光线的分类和组合

CSG 树是一棵二叉树，每个内部节点包含一棵左子树、一棵右子树，以及一个组合运算符（如并、交和差）。因此我们可以表示成如下结构：

```
typedef struct _csgtree{
  Operator op; //defines the combination operator at this node - 0
  for a primitive
  Quadric primitive; //the primitive if this is a leaf node - 0 for
  an interior node
  struct_csgtree *left;
  struct_csgtree *right;
} CSGTree;
```

每个叶节点表示一个基本体素，典型地为一种二次实体。对于任何给定光线，它与 CSG 树中对象的相交计算结果将是一个参数化t值序列t_1、t_2、…，t_n，这里n是相交的数目。然而，为了处理 CSG 树中组合体素，树将依照下列递归算法进行遍历，然后t值按下面描述的方式合并。

```
Classification RayCast(Ray ray, CSGTRee *solid)
{
  if(solid->op){
    left = RayCast(ray,solid->left);
    right = RayCast(ray,solid->right);
    Combine(solid->op,left,right);
  }
  else{
    //transform ray to local primitive coordinates
    switch(solid->primitive){
      CASE cube:........................
      CASE sphere:{intersection
      CASE cone:calculations}
      CASE cylinder:........................
      //etc........................
    }
  }
}
```

380

函数RayCast以光线和表达实体的CSG树指针为参数，返回一个表示相交点的t值序列。（与每个t值相连一个指向记录的指针，该记录是t值所对应的曲面的属性。）如果指针指向一个内部节点或根节点，那么递归地对该节点的左子树和右子树调用该函数。然后将这些递归调用的结果组合起来生成树的总的序列。如果指针是指向一个体素，那么就执行对体素的相交计算。

Roth的文章描述了一个一致的实体表示方法，它有助于相交计算。每个体素在局部坐标系统中表示，举例来说，单位立方体表示方块，位于原点处的单位球体表示球体。现在当系统用户向WC空间中插入一个实体的时候，系统就将其存储为从局部坐标系到 WC 空间的一个4×4变换矩阵（同时还存储这个矩阵的逆矩阵）。当一条光线将与体素相交的时候，只有光线需要被转换到体素所在的局部坐标空间（通过使用逆矩阵），所以所有的相交计算能在局部

空间中进行。这也避免了将实体转换到WC坐标所需要的计算。因此系统对所有体素当前的平移、旋转和缩放存储，完全是通过对矩阵及其逆的存储完成的。

函数`Combine`根据所使用的集合操作产生对t值的分类。这由图18-4 说明。

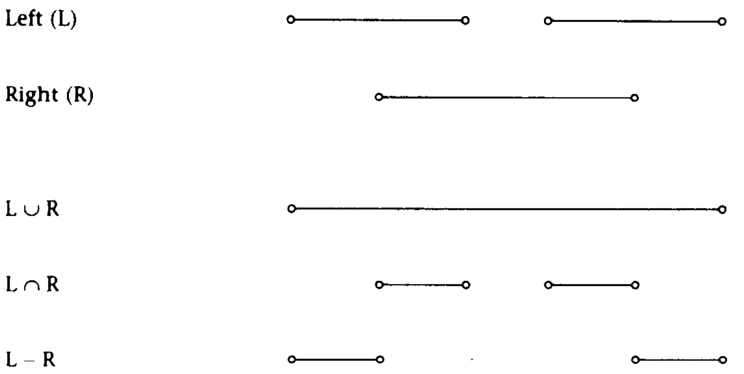


图18-4 CSG组合函数

给定两组t值，它们根据所示的操作算子进行组合，对于三种算子每一个都有明确的解释。

381 光线投射很显然是一种非常费时的计算。Roth 估计超过 50% 的处理时间要花在相交计算上。对于完全的光线跟踪这可能要高达90%。为了要减少计算负荷，Roth 提议对每个体素使用包围盒，以便构造“琐细的”接受/排除测试，如同我们在第16章中所讨论的一样。在那一章中所讨论的各种方法也可以应用到这里来。

18.4 小结

本章主要通过说明球面是更一般的体素——二次曲面的一种特殊情况，进一步扩展了第 5 章的内容。对所有二次曲面都有一个规范的表示方法，我们分析了二次曲面范畴内的一类体素，也包括平面。其次介绍了由基于集合论的体素组合来构造对象的新方式。我们也看到了该如何向这种 CSG 对象投射光线。在这一章中还研究了对象作为实体的性质，而不是只用边界来表示的。注意一个CSG场景可以根本无需多边形进行渲染（使用光线投射）。另一种方法可以参看Salesin and Stofi（1990）。

在下一章中我们转向边界形式表示和二次曲面的参数化表示，而非隐式方程表示。我们

382 将介绍B样条曲线和曲面的重要主题，并表明它们是如何用线段（对曲线）和多边形（对曲面）进行渲染的。

第19章 计算机辅助几何设计介绍

19.1 引言

计算机辅助几何设计（CAGD）是关于曲线和曲面的定义、属性及其渲染的技术，曲线和曲面是很多产品的计算机设计中所不可缺少的。这个领域最初是起源于汽车和造船工业——在这个领域最著名的两位研究者分别是在法国雷诺汽车公司和雪铁龙汽车公司工作的时候进行了开创性的工作。

几何建模的简要历史可以参考 Mortenson（1985，pp.5-7）、Forrest的介绍性评述以及 383 Bartels et al.（1987）、Bézier也给出了进一步的信息，参见Farin（1996）（也可以参见Farin 的指导材料，1992）。他们追溯了CAGD 的源头，即制造业中数控机床（NC）的问题，尤其是在船舶、飞机和汽车工业中。在NC描述中的局限及在统计中曲线和曲面的拟合问题刺激了对曲线、雕刻曲面以及实体的研究，从而产生了放样、Coons面片以及Bézier和 de Casteljau 所定义的三角化和直纹表面面片的工作。

在这一章中我们介绍 CAGD 的基本思想，使用基于*极化形式*（或*开花*）的数学方法。在 CAGD 中使用的曲线和曲面表示成分段多项式的形式，在连接处添加连续性约束。极化形式首先由 de Casteljau（1986）使用的，后来又由Ramshaw（Ramshaw，1987a，1987b和Lee，1989）用来作为表示多项式的方式，很好地展示了它们内在的与Bézier曲线的关系。Ramshaw 为极化形式使用了“开花”这个术语，我们将它与“极化形式”混合使用。我们将在下一小节中介绍多项式和开花，然后使用开花来定义Bézier曲线，以便定义单个多项式曲线段。现实的设计需要许多曲线段端点与端点连接在一起来构造更复杂形状的曲线，这种曲线要比用较低阶（通常不超过3次）的多项式所构造的曲线要复杂得多。但是，当曲线相连时，必须对连接点施加连续性约束。连续性在B样条曲线的构造中讨论。我们将会看到这些思想很容易扩展到曲面。

19.2 多项式和开花

多项式

在这一小节中我们介绍下列思想——函数（或映射）、仿射插值、多项式和它们的开花。多项式对 CAGD而言是基本的，虽然多项式表达式只包括加法和乘法，但是可以用来表示大量的曲线和曲面形状，正如我们将看到的。

多项式具有如下的表达式形式：

$$a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{19-1}$$ 384

这称为n阶多项式，因为t的最高阶为n。举例来说，$1+2t+3t^2$是二阶多项式，$4+5t+6t^2+9t^3$是三阶多项式。

值得注意的是，任意阶的多项式也可以认为是较高阶多项式的“退化”。举例来说，虽然

$1+2t+3t^2$是一个二阶多项式，但是在某种情况下可以视它为退化三阶多项式：$1+2t+3t^2+0t^3$。

我们也将看到多项式是更丰富实体（被称为开花）的一个特殊情况。正是多项式与开花的这种关系允许我们将CAGD理论和方法用相对简单和优美的方式表达。

函数的改编和仿射映射

一个函数是一条规则，即一给定集合（即所谓的域）中每个元素与另一个集合（称为像或值域）中惟一元素的关联。（域和像在某些情况下可能是相同集合。）举例来说，考虑函数x，假设给定任何数t，产生t^2。那么

$$x(t)=t^2 \tag{19-2}$$

假设y是另一个函数，它将任意数加1，那么

$$y(t)=1+t \tag{19-3}$$

最后考虑

$$\begin{aligned}p(t)&=(x(t),\ y(t))\\&=(t^2,\ t)\end{aligned} \tag{19-4}$$

那么p是一个函数，将任何数t映射到二维空间中的一个点。

函数如在式（19-4）中的p，是参数化方程的一个例子。想像$p(t)$是定义在t时刻点的位置，那么当t在某个范围内改变时，点将会在二维空间中一条路径上按照式（19-4）所定义的规则移动。如果读者勾画出t在范围0到1中p的路径会是有益的。

我们在第2章中已经介绍了仿射映射的思想。考虑下面的仿射函数：

$$f(t)=a+bt \tag{19-5}$$

举例来说，$f(t)$可能表示的是一个对象到给定点的距离，当以固定的速度b移动且起始距离

385 为a的时候。现在假设在两个特别的点t_1和t_2上的距离是已知的——换句话说，$f(t_1)$和$f(t_2)$已知。我们需要知道在其他某个时间t的距离。

当然，可以通过使用式（19-5）求出$f(t)$，但是我们改为考虑另一条路径。

下列等式很容易证明：

$$t=\left(\frac{t_2-t}{t_2-t_1}\right)t_1+\left(\frac{t-t_1}{t_2-t_1}\right)t_2 \tag{19-6}$$

该公式表达t是t_1和t_2的加权平均（事实上是重心组合）。

在式（19-5）中的f是一个把实直线R映射到它自身的仿射函数，我们从第8章知道仿射函数保持重心组合。它遵循：

$$f(t)=\left(\frac{t_2-t}{t_2-t_1}\right)f(t_1)+\left(\frac{t-t_1}{t_2-t_1}\right)f(t_2) \tag{19-7}$$

式（19-6）和式（19-7）给了我们一个基本结果，我们将在整章中使用这个结果。

多仿射映射

观察下面这个函数：

$$f(t_1,t_2)=1+2t_1+3t_2+7t_1t_2 \tag{19-8}$$

举例来说，$f(4, 5)=1+2\times4+3\times5+7\times4\times5=164$。假设$t_1$是一个固定值，比如$t_1=1$。那么$f(1, t_2)=1+2+3t_2+7t_2=3+10t_2$，这显然有与式（19-5）相同的形式。同样地，对$t_2$的任何固定值，式（19-8）变成$t_1$的一个仿射函数。方程如式（19-9）称为$t_1$和$t_2$的多仿射函数，因为它是每一个单独考虑的变量的仿射函数。

这里还有更多的例子：

$$f(t_1,t_2)=4+5t_1+4t_2-t_1t_2$$
$$f(t_1,t_2,t_3)=1+3t_1-2t_2+6t_3+t_1t_2-t_1t_3+10t_2t_3-24t_1t_2t_3$$

第二种情况是三个变量的多仿射函数。

通常，n个变量t_1、t_2、…，t_n的多仿射函数是包含下列内容的和项：

- 一个常数（独立于t）；
- 常数与每一个独立变量之积的和（例如$3t_1-2t_2+6t_3$）；
- 每次取两个变量与常数的乘积之和（例如$t_1t_2-t_1t_3+10t_2t_3$）；
- 每次取三个变量与常数的乘积之和；
- 常数乘以每一个变量（例如$-24t_1t_2t_3$）。

多仿射映射的性质

仿射性。多仿射映射对每一个变量是单独仿射的。

对称性。当函数中独立变量的任何置换所得到的函数与原函数有相同的值，则称多仿射映射有对称性。 386

考虑一般的3变量多仿射映射：

$$f(t_1,t_2,t_3)=c_0+c_1t_1+c_2t_2+c_3t_3+c_4t_1t_2+c_5t_1t_3+c_6t_2t_3+c_7t_1t_2t_3 \tag{19-9}$$

为了得到对称性，一定有$c_1=c_2=c_3$，以及$c_4=c_5=c_6$。

这是需求$f(t_1, t_2, t_3)=f(t_i, t_j, t_k)$的结果，这里$(i, j, k)$依次是六个置换(1，2，3)，(1，3，2)，(3，1，2)，(3，2，1)，(2，3，1)，(2，1，3)之一。

对角线性。多仿射映射的对角线性是通过设定所有独立变量为相同值比如t获得的。n变量多仿射映射的对角线性是阶为n的多项式。

举例来说，考虑 4 个变量的多仿射映射：

$$f(t_1,t_2,t_3,t_4)=1+t_1+t_1t_2+5t_2t_3t_4-11t_1t_2t_3t_4 \tag{19-10}$$

那么对角线性是4 阶多项式：

$$f(t,t,t,t)=1+t+t^2+5t^3-11t^4 \tag{19-11}$$

开花定理

多仿射映射与多项式之间存在着很强的关联关系，而多项式是 CAGD 的核心内容：

- 每个n变量多仿射映射有惟一一个n阶多项式作为它的对角线。
- 每个n阶多项式对应于惟一一个对称的n变量多仿射映射，将多项式作为它的对角线。

定理的第一部分是显然的，第二部分不是很明显但是容易证明。我们可以看一个例子。

考虑多项式:

$$F(t)=1+3t+9t^2+5t^3 \tag{19-12}$$

现在任何3变量多仿射映射的对角线是一个三阶多项式。因此以式（19-12）为其对角线的多仿射映射一定有一般形式如式（19-9）。

这个表达式中有八个未知量，c_0到c_7。由对称性我们知道$c_1=c_2=c_3=A$（假设），$c_4=c_5=c_6=B$（假设），所以式（19-9）变成:

$$f(t_1,t_2,t_3)=c_0+A(t_1+t_2+t_3)+B(t_1t_2+t_2t_3+t_1t_3)+c_7t_1t_2t_3 \tag{19-13}$$

现在设$t_1=t_2=t_3$，

387 $$f(t,t,t)=c_0+3At+3Bt^2+c_7t^3 \tag{19-14}$$

将式（19-12）和式（19-14）中系数设为相等，我们得到:

$$\begin{aligned}&c_0=1\\&3A=3,\ 即A=1\\&3B=9,\ 即B=3\\&c_7=5\end{aligned} \tag{19-15}$$

没有其他对称的3变量仿射映射对应于式（19-12）。对于n阶多项式的一般情况，我们可以沿着相似变量，求出多个未知量（c_i）精确匹配由这些未知量表示的多个线性方程，得到惟一的n变量对称的多仿射映射。

这个多仿射映射被称为多项式的开花（或极化形式）。Ramshaw 指出，在某种意义上，开花和多项式表示的是同一个抽象实体；然而，开花是一个更丰富的表示形式。

为了求出任何多项式的开花，我们可以使用下面的规则:

对于一个n阶多项式，第k次幂t^k是从n个独立变量t_1，t_2，…，t_n中选择k项乘积的平均来表示的。举例来说，当$n=3$和$k=2$时有:

$$\frac{t_1t_2+t_1t_3+t_2t_3}{3} \tag{19-16}$$

另外两个例子是:

$$\begin{aligned}F(t)&=1+t+t^2\\f(t_1,t_2)&=1+\frac{t_1+t_2}{2}+t_1t_2\end{aligned} \tag{19-17}$$

$$\begin{aligned}F(t)&=a+bt+ct^2+dt^3\\f(t_1,t_2,t_3)&=a+b\left(\frac{t_1+t_2+t_3}{3}\right)+c\left(\frac{t_1t_2+t_1t_3+t_2t_3}{3}\right)+dt_1t_2t_3\end{aligned} \tag{19-18}$$

19.3 开花和Bézier曲线

de Casteljau 三角形

假设$f(t_1,\ t_2)$是一个多仿射对称性函数，r、s是数且$r<s$。常数x_0、x_1、x_2是给定的，这时:

$$x_0=f(r,r)$$
$$x_1=f(r,s) \qquad (19\text{-}19)$$
$$x_2=f(s,s)$$
388

问题是对任何 $r<t<s$ 求出 $f(t,\ t)$。

首先注意到，求解 $f(t,\ t)$（即对角线），即是在求二阶多项式 $F(t)=f(t,\ t)$（回忆一下多仿射函数的对角线性质）。现在因为 $f(t_1,\ t_2)$ 是多仿射的，它分别对 t_1 和 t_2 仿射的。因此，我们使用式（19-7），求得

$$f(r,t)=\left(\frac{s-t}{s-r}\right)f(r,r)+\left(\frac{t-r}{s-r}\right)f(r,s)$$
$$f(t,s)=\left(\frac{s-t}{s-r}\right)f(r,s)+\left(\frac{t-r}{s-r}\right)f(s,s) \qquad (19\text{-}20)$$

既然 f 假设是对称性的，我们知道 $f(t,\ s)=f(s,\ t)$，因此再次在 r 和 s 之间插值得到：

$$f(t,t)=\left(\frac{s-t}{s-r}\right)f(r,t)+\left(\frac{t-r}{s-r}\right)f(s,t) \qquad (19\text{-}21)$$

来得到所需的值。我们可以用树结构的图形式表示这个计算，如图19-1所示。第一行表示最初的给定值。第二行和导向那一行的边表示式（19-20）。最后一行和导向那一行的边表示式（19-21）。边上的标号是在那些方程中的系数插值。符号表达的一般等值关系在图19-2中示出。这个表示类型称为de Casteljau 三角形。

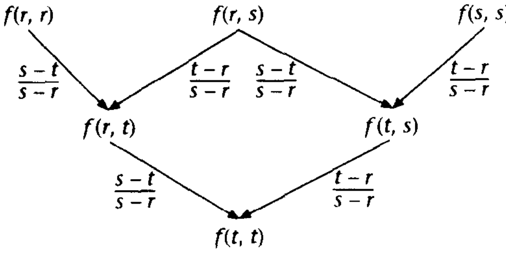


图19-1　对角线计算

相同的技术可以完全被用来解决等价的 3 参数问题：$f(t_1,\ t_2,\ t_3)$ 是对称的多仿射函数，$r<s$，下列值是已知的：

$$x_0=f(r,\ r,\ r)$$
$$x_1=f(r,\ r,\ s)$$
$$x_2=f(r,\ s,\ s) \qquad (19\text{-}22)$$
$$x_3=f(s,\ s,\ s)$$
389

问题是求出 $F(t)=f(t,\ t,\ t)$，这里 t 是在 r 和 s 之间的数。计算可以用图19-3描述。这里使用与先前相同的约定，边的标记是一样的。

读者应该写出类似的4参数的情况，从而得到解 $F(t)=f(t,\ t,\ t,\ t)$。

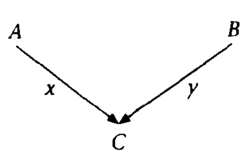


图19-2　插值的图形表示：$C=xA+yB,\ x+y=1$

Bézier曲线

在这一小节中我们介绍CAGD的一个基本形式——Bézier曲线。假设我们给定三个点 $p_0,\ p_1,\ p_2$，以及方程：

$$p_0=f(r,\ r)$$
$$p_1=f(r,\ s) \qquad (19\text{-}23)$$
$$p_2=f(s,\ s)$$

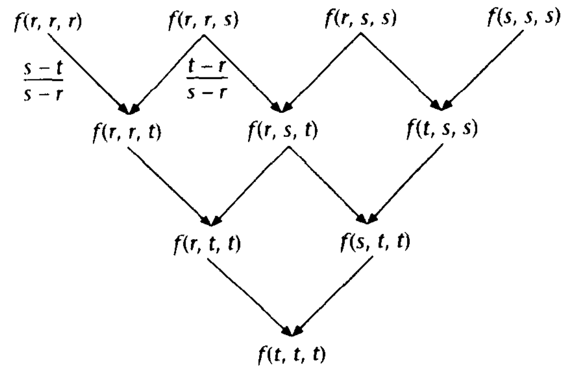


图19-3 计算$F(t)=f(t, t, t)$

现在$f(t_1, t_2)$由两个函数所组成——一个是x单元，一个是y单元：

$$f(t_1,t_2)=(x(t_1,t_2),y(t_1,t_2))$$
$$F(t)=(X(t),Y(t)) \tag{19-24}$$

换句话说，$F(t)$是个取值为向量的多项式，此时包含二个多项式$X(t)$和$Y(t)$，每个都是一个
390 多项式，而且每个都有其对应的极化形式。在本例中，两个多项式是二次的。

现在执行与上相同的插值算法，以便计算对任何t在范围r到s（包含这两个端点），关联于$f(t, t)$的点。这个过程的几何解释如图19-4中所示。

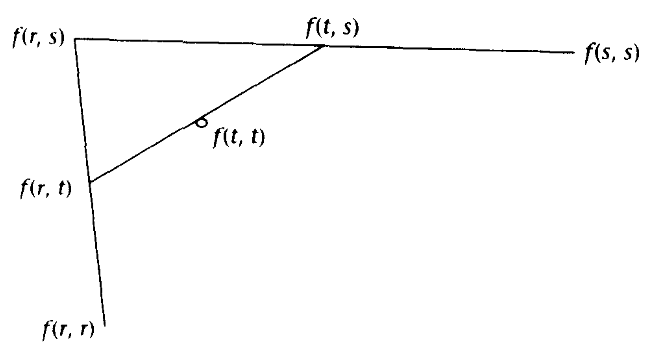


图19-4 de Casteljau三角形的几何解释（二阶）

这个图中的每条边以相同的比率插值，$(s-t):(t-r)$，如式（19-20）和式（19-21）所示。

注意这些插值都是对t的一个特别值进行的。如果我们对在r和s之间的t的每个值重复计算，那么所有这样的点$f(t, t)$会在一条曲线上——即二阶Bézier曲线。初始点p_0，p_1，p_2称为曲线的控制点。

我们可以对3参数情况执行完全相同的过程，如图19-5所示。

这里开始有四个控制点：

$$\begin{aligned} p_0&=f(r, r, r)\\ p_1&=f(r, r, s)\\ p_2&=f(r, s, s)\\ p_3&=f(s, s, s) \end{aligned} \tag{19-25}$$

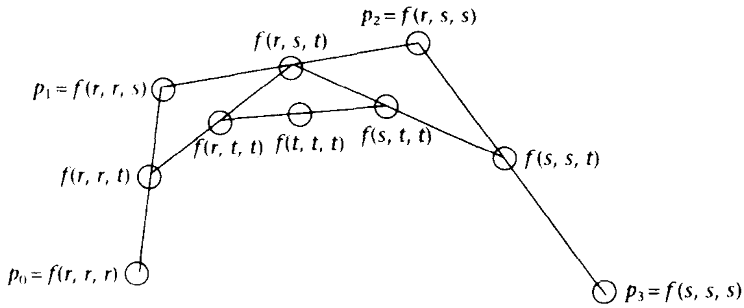


图19-5 计算三次Bézier曲线上的点

我们使用图19-3中显示的算法来求对任何t在r到s范围内的 $f(t, t, t)$值。任何特别的t（例如t=0.5）会给我们曲线上的一个点。如果对所有的t重复这个过程，就会获得三阶Bézier曲线，$t\in[r, s]$。类似可以定义任意阶的Bézier曲线。 391

Bézier曲线的性质

我们能基于曲线的构造方式导出下列各项性质。考虑三次Bézier曲线（三阶），并令曲线上在t处的值为$f(t, t, t)=F(t)$。那么（请读者自己给出这些结果）：

（1）端点插值$F(r)=p_0$和 $F(s)=p_3$。

（2）在参数区间变化情况下的不变性：如果我们把区间$[r, s]$变换到区间$[a+br, a+bs]$（区间的仿射映射），那么曲线形状是不改变的。

（3）凸包性质：曲线上的任何点一定是在控制点定义的凸包里面。

（4）仿射不变性：我们对控制点p_i（i=0，1，2，3）执行仿射变换，并基于这些转换点绘制曲线，那么它与初始曲线上每个点经过相同变换后的曲线是相同的曲线。

（5）如果控制点p_i在一条直线上，那么Bézier曲线是一条直线。

（6）在端点处的曲线切向量是：$F'(r)=3(p_1-p_0)$ $F'(s)=3(p_3-p_2)$

de Casteljau 算法

在图19-5中让我们重新标记如下：

$$\begin{gathered} q_0=f(r,r,r) \quad q_1=f(r,r,t) \quad q_2=f(r,t,t) \\ r_0=q_3=f(t,t,t) \\ r_1=f(t,t,s) \quad r_1=f(t,s,s) \quad r_2=f(s,s,s) \end{gathered} \tag{19-26}$$

如图19-6所示。

检查q点的模式——这给了我们在范围r到t Bézier曲线的控制点，假设称为Q。同样地，r点给了我们Bézier曲线从t到s的控制顶点，假设称为R。这些曲线端点给了我们基于p点的最初的Bézier曲线P。因此得到一个简单的递归产生曲线P的分而治之算法。 392

如果点p_i在一条直线中，绘制这条直线（性质5），否则将这些点分解成Q和R的两个集合，并递归地应用相同的原则到每个点集合。当然，点正好位于一条直线上是不太可能的，但可以取一个误差水平，这样当点在这个误差范围内位于一条直线上时，我们就绘制这条线。

这个算法可以写成:

```
void bezier(Point p[]){
   Point q[], r[];
   if(colinear(p)) Line(p0 ,p3 );
   else{
      /*split p into q and r*/
      split(p,q,r);
      bezier(q);
      bezier(r);
   }
}
```

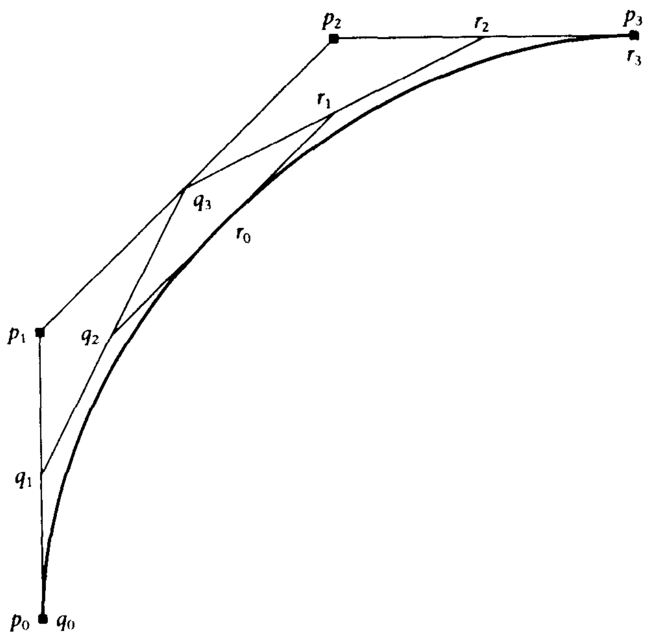


图19-6 三次de Casteljau

当四个点大约在一条直线上时，布尔函数`colinear`返回“真”值，而函数`split`计算在某个适当值如在 t=0.5 处的点，使用与上面相同的算法。

为了要测试四个点是否大约在一条直线上，求连结p_0和p_3的方程，以$ax+by+c=0$的形式。那么任意点(X, Y)与这条直线的距离$D(X, Y)$由下式给出:

$$D^2(X,Y)=\frac{(aX+bY+c)^2}{a^2+b^2} \tag{19-27}$$

因此，如果$D(p_1)$和$D(p_2)$每个都小于某个预先定义的误差值，四个点可以说是大约共线的。
393 不等式$D^2(X, Y)<T$，对于某个误差T，可以被组织成这样的方式，即在求值过程中不涉及除法（通过遍乘分母，它总是正的）。

19.4 Bézier曲线和Bernstein基

Bézier曲线的多项式形式

考虑二阶Bézier曲线，定义如式（19-23）和式（19-24）。不失一般性（性质2），我们限

制参数范围为[0、1]、即$r=0$、$s=1$。那么由式（19-20）:

$$\begin{aligned} f(0,t) &= (1-t)f(0,0)+tf(0,1)=(1-t)p_0+tp_1 \\ f(t,1) &= (1-t)f(0,1)+tf(1,1)=(1-t)p_1+tp_2 \end{aligned} \tag{19-28}$$

因此使用式（19-21）:

$$\begin{aligned} F(t) &= f(t,t)=(1-t)^2 f(0,t)+tf(t,1) \\ &= (1-t)^2 p_0+2t(1-t)p_1+t^2 p_2 \end{aligned} \tag{19-29}$$

我们可以为三次情况构造一个相似的函数变量，从式（19-25）开始。此时:

$$\begin{aligned} F(t) &= f(t,t,t) \\ &= (1-t)^3 p_0+3(1-t)^2 tp_1+3(1-t)t^2 p_2+t^3 p_3 \end{aligned} \tag{19-30}$$

在式（19-35）中点的系数是$((1-t)+t)^2$二项式展开中的连续项、在式（19-36）中是$((1-t)+t)^3$的连续项。通常、如果我们执行对任何n阶的一个相似的求导、就会得到相同的结果。（这可以被证明、比如使用归纳法。）

因此定义一般（n阶）Bézier曲线上的一个点的方法是用$n+1$个控制点、与多仿射函数成为一体:

$$\begin{aligned} p_0 &= f(0,0,\cdots,0) \\ p_1 &= f(0,0,\cdots,1) \\ &\cdots \\ p_n &= f(1,1,\cdots,1) \end{aligned} \tag{19-31}$$

这里、在展开中对于p_i、有i个1和$(n-i)$个0。

如果我们应用规则来求对角线$F(t)=f(t、t、\cdots、t)$、对于任何$t\in[0、1]$、会得到:

$$F(t)=\sum_{i=0}^{n} B_{n,i}(t)p_i, \quad t\in[0,1] \tag{19-32}$$

这里

$$B_{n,i}(t)=\binom{n}{i}t^i(1-t)^{n-i}$$ 394

和

$$\binom{n}{i}=\frac{n!}{i!(n-i)!} \tag{19-33}$$

Bernstein基

函数$B_{n,i}(t)$是著名的Bernstein基函数。基的性质是任何多项式完全可以被惟一表示成基函数的一个线性组合。举例来说，假设我们要把重点放在最高为三阶的多项式、那么任意多项式有形式:

$$a_0+a_1t+a_2t^2+a_3t^3 \tag{19-34}$$

现在单项的（幂）基由多项式1、t、t^2、t^3所组成。显然对任何具有如式（19-34）形式的多项式可以被写成这些基本多项式的仿射组合。同时对任何这种多项式可以写成下列形式:

$$\alpha_0 B_{3,0}(t)+\alpha_1 B_{3,1}(t)+\alpha_2 B_{3,2}(t)+\alpha_3 B_{3,3}(t) \tag{19-35}$$

在这个意义上，这些函数$B_{3,i}(t)$是所有的三阶多项式的一组基。通常n阶Bernstein多项式是所有n阶（或更低的）多项式的一组基。

这些基函数的性质是：

$$\begin{aligned} &B_{n,i}(t)\geqslant 0,\ t\in[0,1] \\ &\sum_{i=0}^{n} B_{n,i}(t)=1 \end{aligned} \tag{19-36}$$

第一个从定义中可以清楚看出，而第二个是因为这些函数$((1-t)+t)^n=1$展开式的连续项。从这些性质及式（19-33）我们能得到如下推论：

- Bézier 曲线上的任何点是控制点的一个重心组合。
- 事实上是一个凸组合，因为系数总和为1。
- 曲线上的任何点一定位于控制点的凸包里面。

切向量

给定一个参数化方程定义的曲线，例如

$$F(t)=(X(t),Y(t))=(1+3t^2-t^3,1+3t-t^3) \tag{19-37}$$

我们会使用传统分析工具研究曲线的某些性质（例如曲线的梯度）。定义切向量为：

395

$$F'(t)=(X'(t),Y'(t)) \tag{19-38}$$

这里斜撇符号代表对参数的微分。这个切向量是一个指示曲线上与t所对应点的方向矢量，切向量的大小说明曲线在那一点上的“平坦”程度。

因此对于上面的曲线，

$$F'(t)=(6t-3t^2,3-3t^2) \tag{19-39}$$

曲线的始点和终点的切向量具有特别的重要性。假设这个曲线是定义在范围 $t\in[0,1]$上，那么在曲线初始点上 $F'(0)=(0,3)$，在终点上 $F'(1)=(3,0)$。这些切向量的方向和长度给出了曲线在两端点处的形状指示。

让我们求出由式（19-37）所定义的曲线的Bézier控制点。使用式（19-18），极化形式是：

$$\begin{aligned} &f(t_1,t_2,t_3)=(x(t_1,t_2,t_3),y(t_1,t_2,t_3)) \\ &x(t_1,t_2,t_3)=1+(t_1t_2+t_2t_3+t_1t_3)-t_1t_2t_3 \\ &y(t_1,t_2,t_3)=1+(t_1+t_2+t_3)-t_1t_2t_3 \end{aligned}$$

将（t_1、t_2、t_3）分别用（0、0、0），（0、0、1），（0、1、1）和（1、1、1）替换，我们有：

$$\begin{aligned} p_0&=(1,\ 1) \\ p_1&=(1,\ 2) \\ p_2&=(2,\ 3) \\ p_3&=(3,\ 3) \end{aligned}$$

使用Bézier 曲线的性质以及切向量，曲线（在图19-7中示出的）的形状是可以确定的。

考虑由式（19-30）所定义的三次曲线。如果我们将表达式对t求微分，并求t=0和t=1处的结果，得到：

$$F'(0)=3(p_1-p_0)$$
$$F'(1)=3(p_3-p_2) \tag{19-40}$$

换句话说，在曲线的始端和终端，切向量的方向分别与由控制点p_0和p_3所定义的多边形控制图的始边和终边方向相同。

这是一般性的结论，很容易证明对于n阶Bézier曲线（式（19-32））：

$$F'(0)=n(p_1-p_0)$$
$$F'(1)=n(p_3-p_2) \tag{19-41}$$

这个性质对于辅助设计者迅速地掌握控制点和最后曲线形状之间的关系是非常重要的。

$$F^r(t)=\frac{n!}{(n-r)!}\sum_{i=0}^{n-r}\Delta^r p_i B_{n-r,i}(t) \tag{19-42}$$ 396

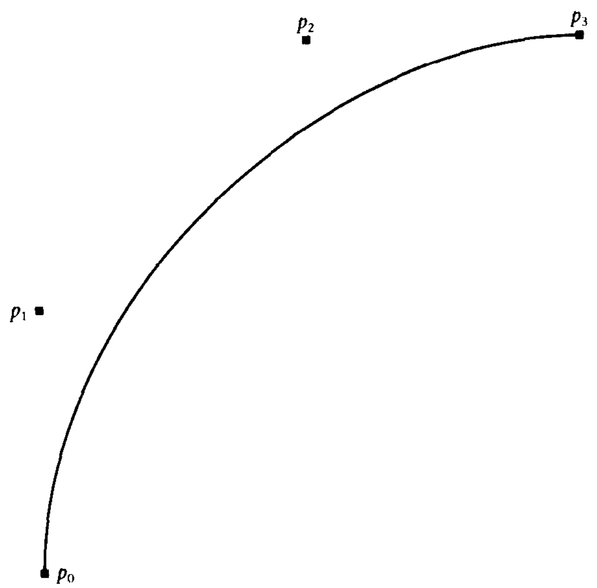


图19-7 式（19-37）对应的Bézier曲线

这里Δ是前向差分算子：

$$\Delta^0 p_i \equiv p_i$$
$$\Delta p_i = p_{i+1}-p_i$$
$$\Delta^r p_i = \Delta^{r-1}p_{i+1}-\Delta^{r-1}p_i, r=1,2,3,\cdots \tag{19-43}$$

因此Bézier曲线的导数是由对最初点的前向差分所形成的Bézier曲线（注意控制点在这种情况下是矢量）。

19.5 升阶Bézier曲线

使用退化多项式

一个n阶Bézier曲线能有最多$(n-1)$个“拐点”——比如最大值和最小值。在曲线设计中

为了得到更大的灵活性，可以使用的一种办法是通过将多个相对低阶的曲线首尾相连，这样就获得了在这些点处的连续性。另一种方法是从一个低阶曲线开始，求出一组新的控制点来
397 描述相同的曲线，但是有比较高的阶。这是表达任何给定的多项式为"退化"的更高阶多项式的惟一方式：退化在某种意义上说就是高次幂的系数为零。举例来说，多项式$1+2t+3t^2$可以被看成是三次多项式，只是t^3的系数正好为零而已。它可以被看作为任意阶多项式，三阶或是更高的，只要相关的系数为零就行。

开花和退化多项式

考虑多项式

$$F(t)=a_0+a_1t+a_2t^2 \tag{19-44}$$

它的开花是

$$f(t_1,t_2)=a_0+a_1\left(\frac{t_1+t_2}{2}\right)+a_2t_1t_2 \tag{19-45}$$

现在把式（19-44）看成是"退化"的三次多项式。那么此时的开花是：

$$g(t_1,t_2,t_3)=a_0+a_1\left(\frac{t_1+t_2+t_3}{3}\right)+a_2\left(\frac{t_1t_2+t_1t_3+t_2t_3}{3}\right) \tag{19-46}$$

我们把它留给读者作为一个练习，很容易证明在式（19-45）和式（19-46）之间存在一个关系，事实上有：

$$g(t_1,t_2,t_3)=\frac{f(t_1,t_2)+f(t_1,t_3)+f(t_2,t_3)}{3} \tag{19-47}$$

从第一条法则，我们知道f是F的开花，所以f是对称的且是多仿射的，它的对角线是F。因此g一定也是对称的和多仿射的，而且它的对角线也是F。所以g一定是F的惟一极化形式，当F被当成是三次多项式的时候。

Bézier 曲线的应用

假设我们有一个基于控制点p_0、p_1和p_2的二次Bézier曲线。如果将参数范围设为$t\in[0,1]$，

$$\begin{aligned}p_0&=f(0,0)\\p_1&=f(0,1)\\p_2&=f(1,1)\end{aligned} \tag{19-48}$$

这里f是曲线的开花。我们希望将这条曲线表达成三次Bézier曲线，那么对应的控制点应该是什么？假设这些控制点是

398 $$q_0,\ q_1,\ q_2,\ q_3 \tag{19-49}$$

我们从Bézier曲线的端点条件性质中知道有$q_0=p_0$和$q_3=p_2$，那么另外的一些点是什么呢？假设对应的三次多项式的开花是$g(t_1,t_2,t_3)$。那么

$$\begin{aligned}q_0&=g(0,0,0)\\q_1&=g(0,0,1)\\q_2&=g(0,1,1)\\q_3&=g(1,1,1)\end{aligned} \tag{19-50}$$

由式（19-47）：

$$\begin{aligned} q_0 &= g(0,0,0) = f(0,0) = p_0 \\ q_1 &= g(0,0,1) = \frac{1}{3}f(0,0) + \frac{2}{3}f(0,1) = \frac{1}{3}p_0 + \frac{2}{3}p_1 \\ q_2 &= g(0,1,1) = \frac{2}{3}f(0,1) + \frac{1}{3}f(1,1) = \frac{2}{3}p_1 + \frac{1}{3}p_2 \\ q_3 &= g(1,1,1) = f(1,1) = p_2 \end{aligned} \tag{19-51}$$

新的点因此可以从旧的点中计算出来。

一般性结果

假设我们从$n+1$个控制点得到一个Bézier曲线，需要得到相同Bézier曲线的$n+2$个控制点。假设对于$n+1$情况的极化形式是$f(t_1, t_2, \cdots, t)$，而对于$n+2$情况是$g(t_1, t_2, t_{n+1})$，

$$g(t_1, t_2, \cdots, t_{n+1}) = \frac{f(t_1, t_2, \cdots, t_n) + f(t_1, t_2, \cdots, t_{n-1}, t_{n+1}) + \cdots + f(t_2, t_3, \cdots, t_{n+1})}{n+1} \tag{19-52}$$

这里和式中每一项都缺一个t_i参数。这个结果看起来是正确的，因为右侧的表达式是对称的和多仿射的，$g(t, t, \cdots, t) = f(t, t, \cdots, t)$，而且极化形式是惟一的。

假设最初的控制点是p_i，而新的控制点是q_i。那么，

$$q_i = g(\underbrace{0,0,\cdots,0}_{n-i+1},1,1,\cdots,1) \tag{19-53}$$

这里有$n+1-i$个0和i个1。将它代入g的表达式中有：

$$q_i = \frac{(n-i+1)p_{i-1} + ip_i}{n+1} = \left(1 - \frac{i}{n+1}\right)p_{i-1} + \left(\frac{i}{n+1}\right)p_i \tag{19-54}$$ 399

19.6 有理Bézier曲线

介绍

Bézier曲线为曲线设计提供了一个灵活的方法。然而，有一些重要的曲线在CAGD中十分有用却不能用Bézier曲线精确表示出来，无论使用多么高的阶。圆锥截面曲线就是这种情况，一个圆就无法用它表示。Bézier曲线在仿射变换下是不变的（无论是对控制点进行转换然后渲染由这些转换的控制点表示的曲线，还是对初始的曲线本身作变换，我们都能得到相同的曲线），但是它们在投影变换下是变化的。

举例来说，假设Bézier曲线控制点(x_i, y_i, z_i)，$i=0,\cdots,n$，视平面是XY平面，投影中心在$(0, 0, -1)$，观察方向沿Z轴方向，那么这些点的投影是在

$$\left(\frac{x_i}{z_i+1}, \frac{y_i}{z_i+1}, 0\right) \tag{19-55}$$

现在如果Bézier曲线是从这些点构造的，很明显所得到的曲线与通过投影初始曲线上的点$(x(t), y(t), z(t))$，$t \in [0, 1]$所得到的曲线不是同一条曲线。读者可以证明这种情况的确存在。

通常，有理多项式是两个多项式的比。比如：

$$\frac{a_0+a_1t+a_2t^2+\cdots+a_nt^n}{b_0+b_1t+b_2t^2+\cdots+b_nt^n} \tag{19-56}$$

一个有理参数化曲线的$x(t)$、$y(t)$表达式是有理多项式（如果是空间曲线的话，还有$z(t)$）。

考虑：

$$\begin{aligned} x(t)&=\frac{1-t^2}{1+t^2} \\ y(t)&=\frac{2t}{1+t^2} \\ t&\in[0,1] \end{aligned} \tag{19-57}$$

容易验证有$x(t)^2+y(t)^2=1$，说明这个参数化曲线表示的是四分之一圆周。这样二阶有理参数化曲线能精确地表示出一个圆，而任何有限阶的非有理曲线是不能精确表示一个圆的。

有理Bézier曲线

为了定义一个有理Bézier曲线，我们在每个点上附上一个数$w_i>0$作为权值。给定控制点$(x_i，y_i)$，我们以齐次形式将这些点表示成$(w_ix_i，w_iy_i，w_i)$（如果是三维曲线，$(x_i，y_i，z_i)$
400 $=(w_ix_i，w_iy_i，w_iz_i，w_i)$）。

我们知道齐次点$(w_ix_i，w_iy_i，w_i)$一般等价于一个二维点$(x_i，y_i)$，这总是通过除以最后一项（权值）得到的。

现在我们像通常那样构造 Bézier曲线，在齐次空间中表示形式为$(x(t)，y(t)，w(t))$，基于控制点$(w_ix_i，w_iy_i，w_i)$，在二维空间中曲线上的真实点是通过用最后一项除来得到的。因此：

$$\begin{aligned} X(t)&=\frac{x(t)}{w(t)}=\frac{\sum_{i=0}^{n}B_{n,i}(t)w_ix_i}{\sum_{i=0}^{n}B_{n,i}(t)w_i} \\ Y(t)&=\frac{y(t)}{w(t)}=\frac{\sum_{i=0}^{n}B_{n,i}(t)w_iy_i}{\sum_{i=0}^{n}B_{n,i}(t)w_i} \end{aligned} \tag{19-58}$$

注意当所有的w_i都相等的时候，式（19-58）变成一个非有理 Bézier 曲线，因为w_i消掉了，分母对所有的t都等于1。

有理曲线的求解

一条有理Bézier曲线可以通过我们平常处理Bézier曲线的方法来求解。只是在最后一步，x坐标和y坐标每个都要除以w坐标。举例来说，考虑二次有理曲线，其控制点为（0，0）、（1，1）和（2，0），其对应的权值分别是1、2和 3。那么构造齐次控制点（0，0，1）、（2，2，2）和（6，0，3）。现在让我们在t值为0.5处求解这条曲线，使用细分（de Casteljau 算法）（表19-1）。

最后的点在齐次空间中是(2.5、1、2)、(2.5/2、1/2、1)≡(1.25、0.5)。

表19-1 求解一条有理 Bézier曲线 401

控制点	中点	中点
(0,0,1)		
	(1,1,1.5)	
(2,2,2)		(2.5,1,2)
	(4,1,2.5)	
(6,0,3)		

同样地，前向差分方法（见后面）可以用来求解在$(x(t)、y(t))$和$w(t)$上的点，比值在最后一步计算出来。

从一般形式转换到Bézier形式

考虑式（19-57），它表示的是四分之一圆，参数范围是[0，1]。假设我们希望求得它的控制点和权值，表示成Bézier形式。齐次表示为$(x(t)、y(t)、w(t))=(1-t^2,\ 2t,\ 1+t^2)=F(t)$。那么求每个二次方程的开花，所以等价的开花函数是：

$$f(t_1,t_2)=(1-t_1t_2,t_1+t_2,1+t_1t_2) \qquad (19\text{-}59)$$

控制点是$f(0, 0)$、$f(0, 1)$和$f(1, 1)$，有

$$f(0, 0)=(1, 0, 1)$$
$$f(0, 1)=(1, 1, 1) \qquad (19\text{-}60)$$
$$f(1, 1)=(0, 2, 2)$$

现在将这些转换到二维空间，获得控制点（1，0）、（1，1）和（0，1），其权值分别是1，1和2。读者可以利用这些思想写出一个程序来生成一个完整的圆。

性质

通常，越是较大的权值附给控制点，与其他权值相比，那个控制点的影响就越大。因此权值可以用来交互地调整曲线的形状（比如在每个控制点上提供一个滑杆，允许用户调整权值）。

有理Bézier曲线共享非有理曲线的仿射变换不变性，它们在投影变换下也是不变的。

19.7 曲线的拼接：连续性

分段多项式曲线段

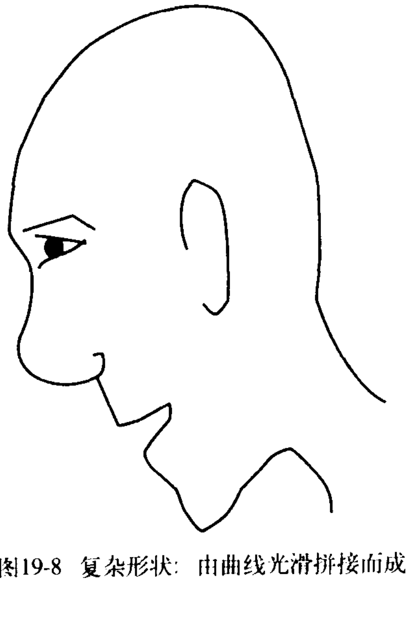


图19-8 复杂形状：由曲线光滑拼接而成

为了设计复杂形状（如图19-8），我们必须使用非常高阶的多项式，或者利用曲线段的拼接构造。提高阶次来增加复杂性最终会弄巧成拙，因为它是典型不实用的，而且使用高阶曲线没有什么意义： 402

• 多项式变得在数值上不稳定。
• 与较低阶曲线相比，计算开支增加太快。
• 总体形状没有局部控制特性——即不可能改变曲线某个部分而不造成整个曲线的变化。

另一种方法是拼接多条低阶曲线形成一条曲线，用这种方法某些连续性需求在那些结合处得到满足。我们介绍参数化连续性的概念，通过这种思想两曲线可以连接在一起。

参数化连续性

假设$F(t)$，$t\in[t_0, t_1]$和$G(t)$，$t\in[t_1, t_2]$是两个k阶多项式，那么我们说$F(t)$和$G(t)$ 在t_1具有C^0连续性，如果$F(t_1)=G(t_1)$。也就是说，它们只是在参数 $t=t_1$处相连。C^1连续性需要它们在$t=t_1$处有相同的一阶导数，即 $F'(t_1)= G'(t_1)$。一般来讲，C^r连续性需要直到r阶导数并且包括r阶
403 导数在内的所有导数在 $t=t_1$处有$F^r(t_1)=G^r(t_1)$。曲线连接处的参数值通常叫做节点。

在CAGD的上下文中，我们感兴趣的是参数化定义的曲线形式 $F(t)=(X_F(t), Y_F(t))$和$G(t)=(X_G(t), Y_G(t))$在节点$t=t_1$处的连续性。结果是一样的，只是现在我们有$F^r(t)=(X_F^r(t), Y_F^r(t))$。

因此当$r=1$时，$F^r(t)$是切向量，表示点（$X_F(t)$，$Y_F(t)$）沿着曲线在 t处的变化率，当t为时间时，这可以解释成速度。同样地，当$r=2$，此时等价于加速度。

虽然给定的两个多项式相对简单，但是保证它们能满足必须的连续性是件困难的事情。问题总是归结为求解一组线性方程。举例来说：

$$F(t)= a_0+a_1t+a_2t^2+a_3t^3$$
$$G(t)= b_0+b_1t+b_2t^2+b_3t^3 \tag{19-61}$$

假设多项式F是已知的（即系数a_i是已知的），我们需要G在$t=1$处与F连接，且有C^2连续性。那么，我们需要：

$$\begin{aligned}F(1)=G(1)&\Rightarrow a_0+a_1+a_2+a_3=b_0+b_1+b_2+b_3\\F'(1)=G'(1)&\Rightarrow a_1+2a_2+3a_3=b_1+2b_2+3b_3\\F^2(1)=G^2(1)&=2a_2+6a_3=2b_2+6b_3\end{aligned} \tag{19-62}$$

这里有四个未知数b_0、…、b_3和三个方程，所以需求附加约束条件到系数b_i的一个自由度上。

注意，对于三阶多项式，C^3连续性需要G与F是完全相同的多项式（因为让三阶导数相等有$a_3=b_3$，那么代入到二阶需求得到 $b_2=a_2$，所以 $b_i=a_i$，对所有的i）。通常，对于k阶多项式，我们惟一感兴趣的是直到 C^{k-1}连续性。

几何连续性

在这一小节中我们证明参数化连续性是一个非常严格的要求。假设$P(t)$和$Q(t)$是两个k阶Bézier 曲线，分别定义在范围 [0，1] 和 [1，2]上。我们需要在 $t=1$处有C^1连续性，那么我们知道在曲线的始端和终端有：

$$P'(1)=k(p_k-p_{k-1})=Q'(1)=k(q_k-q_{k-1}) \tag{19-63}$$

我们也需要：

$$P(1)=p_k=Q(1)=q_0 \tag{19-64}$$

从这里我们导出 C^1连续性：

$$
\begin{aligned}
q_0 &= p_k \\
q_1 &= 2p_k - p_{k-1}
\end{aligned}
\tag{19-65}
$$

因此，参数化连续性需要q_1一定位于由控制点p_{k-1}和p_k所定义的直线上一个确切的距离处。 404
然而我们知道，如果q_1是沿着这条直线在任何距离处，两曲线之间会有一阶视觉连续性。参数化连续性看起来是十分精确和严格的一种要求。G^1（几何的）连续性是能得到满足的，假若q_1在适当的直线上；从连续性的角度去看，实际的距离不是我们所关心的，虽然它会影响曲线的总体形状。

尽管参数化连续性是一个比我们通常直觉理解的“连续性”更严格的要求，但是从数学处理上看实际它比几何连续性要容易处理。这里我们只关注参数化连续性。

对开花求导

假设$F(t)$是一个k阶多项式，$f(t_1, \cdots, t_k)$是它的极化形式，那么我们表达F的k阶导数为：

$$
F^r(t) = \frac{k!}{(k-r)!}\cdot\sum_{i=0}^{r}\binom{r}{i}(-1)^{r-i} f(t^{\langle k-i\rangle},(t+1)^{\langle i\rangle})
\tag{19-66}
$$

这里我们使用符号$t^{\langle k-i\rangle}$、$(t+1)^{\langle i\rangle}$来表示序列$t, t, \cdots, t, (t+1), \cdots, (t+1)$，有连续$k-i$个$t$和$i$个$(t+1)$。

这个公式看起来很复杂，但可以通过一个简单的例子说明它。

假设$F(t)=t^3$，那么极化形式是$f(t_1, t_2, t_3)=t_1t_2t_3$。现在使用公式求解$F'(t)$。这里$k=3$，$r=1$。那么：

$$
\begin{aligned}
F'(t) &= \frac{3!}{2!}\cdot(-f(t,t,t)+f(t,t,t+1)) \\
&= 3(-t^3+t^2(t+1)) \\
&= 3t^2
\end{aligned}
\tag{19-67}
$$

当然，在类似这种情况下是很容易直接求出导数的，但是稍后我们将会看到一些应用，这个结果在那里是非常有用的。大致的证明如下。

证明。考虑某个a和t'，表达式

$$
F(t+a(t'-t)) = f(t+a(t'-t), t+a(t'-t), \cdots, t+a(t'-t))
\tag{19-68}
$$

现在

$$
t+a(t'-t) = (1-a)t+at'
\tag{19-69}
$$

因此，使用多仿射性质：

$$
\begin{aligned}
& f((t+a(t'-t), \cdots, t+a(t'-t)) = (1-a)f(t, t+a(t'-t), \cdots, \\
& t+a(t'-t)+af(t', t+a(t'-t), \cdots, t+a(t'-t))
\end{aligned}
\tag{19-70}
$$

继续使用多仿射性质来展开各项，最后得到（使用归纳法）： 405

$$
F(t+a(t'-t)) = \sum_{i=0}^{k}\binom{k}{i}a^i(1-a)^{k-i}f(t^{\langle k-i\rangle}, t'^{\langle i\rangle})
\tag{19-71}
$$

通过泰勒展开:

$$F(t+a(t'-t))=\sum_{i=0}^{k}\frac{a^i(t'-t)^i}{i!}F^{(i)}(t) \tag{19-72}$$

（所有的k阶之后的导数都是零）。通过扩展式（19-71）中a的幂项$(1-a)^{k-i}$，结果可得。重新整理和式的指数，然后让a 系数相等。特别地，选择$t'=1+t$。

连续性条件

假设我们有两个k阶多项式$F(t)$，$t\in[a，s]$和 $G(t)$，$t\in[s，b]$。那么，在参数值s处，它们有C^r连续性当且仅当

$$f(u_1,u_2,\cdots,u_r,s^{\langle k-r\rangle})=g(u_1,u_2,\cdots,u_r,s^{\langle k-r\rangle}) \tag{19-73}$$

对任何序列u_1、u_2、…、u_r。证明可以参见（Seidel，1989）。注意，它还隐含着这样的内容，即C^r连续性暗示着所有更低阶的连续性，因为u序列当然能包括s的实例。

假设

$$\begin{aligned}F(t)&=t^2\\G(t)&=a_0+2a_1t+a_2t^2\end{aligned} \tag{19-74}$$

分别有域 [0，1] 和 [1，2]。我们将确定在节点t=1 处C^1连续性的条件。

$$\begin{aligned}f(t_1,t_2)&=t_1t_2\\g(t_1,t_2)&=a_0+a_1(t_1+t_2)+a_2t_1t_2\end{aligned} \tag{19-75}$$

在这个例子中，我们有 k=2，s=1。

首先考虑C^0连续性，依照要求有:

$$f(1,1)=g(1,1),\ a_0+2a_1+a_2=1 \tag{19-76}$$

现在考虑C^1连续性。这需要:

$$f(u_1,1)=g(u_1,1) \tag{19-77}$$

因此:

$$u_1=a_0+a_1(u_1+1)+a_2u_1 \tag{19-78}$$

根据式（19-76）有$a_0=1-2a_1-a_2$，将它代入到式（19-78），重新整理得:

406 $$a_1(u_1-1)+a_2(u_1-1)=u_1-1 \tag{19-79}$$

从式（19-79）我们获得需求$a_1+a_2=1$。（注意，我们可以除以 u_1-1，安全地假定$u_1\neq1$，因为如果u_1=1便返回式（19-78）。）

19.8　B样条曲线

分段多项式曲线

在这一小节中我们要说明如何构造k阶分段连续的多项式曲线段，使之在连接点处具有C^{k-1}连续性。这些就是所谓的B样条曲线。我们仅就k=3，三次B样条进行介绍，但是结果可以

般化到任何阶。首先研究一下单个曲线段，给出一些基本结果，然后讲解多个这样的曲线段可以按照需要的参数化连续性连接在一起。

单多项式曲线段

假设我们有标量值序列t_1、t_2、…、t_{2k}。这些被称为节点。我们要求这些值是非降序排列的：

$$\begin{aligned} &t_i \leqslant t_j,\ i<j \\ &t_i < t_{i+k} \end{aligned} \tag{19-80}$$

这些节点值的选取是由曲线设计者决定的。通常它们之间的空间间隔是相等的，例如1，2，3，…，这称为*一致节点序列*。假设v_0、v_1、…，v_k是空间中任意的点序列，那么可以证明（见Ramshaw，1989）有一个惟一的k阶多项式 $F(t)$，对t的开花有

$$\begin{aligned} &v_i=f\,(t_{i+1},\ t_{i+2},\ \cdots,\ t_{i+k}) \\ &i=0,\ 1,\ \cdots,\ k \end{aligned} \tag{19-81}$$

这不难证明。式（19-81）实际上给了我们k+1个线性方程，k+1个未知数，如果我们采用开花的一般形式的话。未知数是系数a_i。表达线性方程的矩阵满秩，因此是可逆的。这对于k=2,3的情形特别容易证明。

取k=2这种情形，所以我们有节点t_1、t_2、t_3、t_4和控制点v_0、v_1、v_2。那么定理指出存在一个惟一的多项式，有开花f如下：

$$\begin{aligned} v_0 &= f(t_1,t_2) = a_0 + a_1\left(\frac{t_1+t_2}{2}\right) + a_2 t_1 t_2 \\ v_1 &= f(t_2,t_3) = a_0 + a_1\left(\frac{t_2+t_3}{2}\right) + a_2 t_2 t_3 \\ v_2 &= f(t_3,t_4) = a_0 + a_1\left(\frac{t_3+t_4}{2}\right) + a_2 t_3 t_4 \end{aligned} \tag{19-82}$$

407

a_i是未知数，但是所有其他值都是已知的。这给了我们三个方程，包含三个未知数。（事实上，因为我们是在处理点，对于二维空间将会有两组这样的方程，而对于三维空间，有三组这样的方程。）

通常，为了求得曲线上的一个点，我们可以使用极化形式的多仿射性质，并不断地通过插值“开花”这些值。由此可以看到有足够多定义好的节点，通过插值，我们可以求出在t_k和t_{k+1}之间的任何值。因此对这个范围内的任何t，我们需要求出f（t，t，…，t）。

举例来说，假设我们有节点1，2，3，4，5，6 和控制点v_i：（0，0），（1，0），（1，1）和（0，1）。我们想要求出曲线上t=3.5处的那一点，那么，通过式（19-81），一定有：

$$\begin{aligned} (0,0)&=f(1,2,3) \\ (1,0)&=f(2,3,4) \\ (1,1)&=f(3,4,5) \\ (0,1)&=f(4,5,6) \end{aligned} \tag{19-83}$$

我们因而可以构造一个 de Casteljau 三角形来执行计算，如图19-9所示。

读者将会发现绘制一个略图说明在控制点上的插值是很有效的方法。这个练习对于在3和4之间的t值要重复多遍，这样可识别出二次曲线上的一些点。

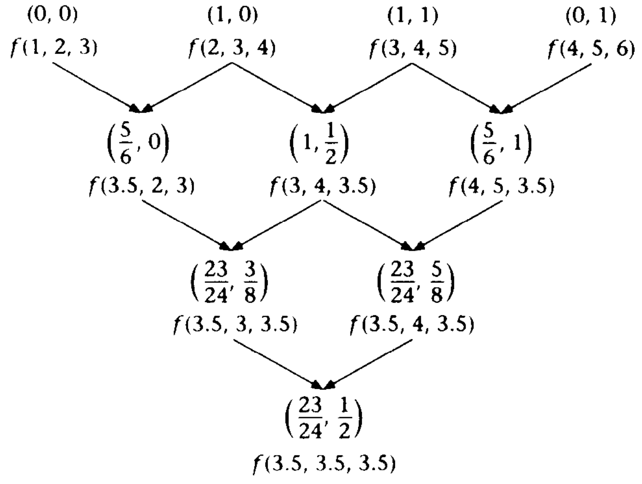


图19-9 式（19-83）所对应的f(3.5、3.5、3.5)的de Casteljau三角形

两多项式曲线段

继续单一曲线段的情况，假设我们增加另一个点v_{k+1}以及另一个节点t_{2k+1}，现在考虑序列t_2、
408 t_3、…、t_{2k+1}。根据前面一小节中的结果，这对应于控制点v_1、…、v_{k+1}以及多项式曲线G及其极化形式，定义在范围$[t_{k+1}, t_{k+2}]$。举例来说，$v_{k+1}= g(t_{k+2}, \cdots, t_{2k+1})$。现在假设希望求得这个曲线上的一个点，$g(t, t, \cdots, t)=G(t)$，$t\in[t_{k+1}, t_{k+2}]$。当然，我们会按一般方式求插值。

取一个特例情况$t=t_{k+1}$。这是在第一条曲线段的结尾和下一条曲线的开始。如果我们求得了$f(t_{k+1}, t_{k+1}, \cdots, t_{k+1})$和$g(t_{k+1}, t_{k+1}, \cdots, t_{k+1})$，通过构造，$f$和$g$的开花值一定是相等的，无论它们分别与de Casteljau三角形重合在何处。因为它们插值于控制点集合v_1、…、v_k，所以一定是相等的。对出现的各种等式的检查说明，f和g开花之间的关系完全满足方程对式（19-73）给出的C^{k-1}连续性的要求。

考虑k=3的情况和两个三角形f和g，那么求解在t_4处的值有如图19-10中所示的三角形。最高一行给出在t_4处C^2连续性的条件。第二行给出C^1连续性条件，而最底下一行给出的是C^0连续性条件。

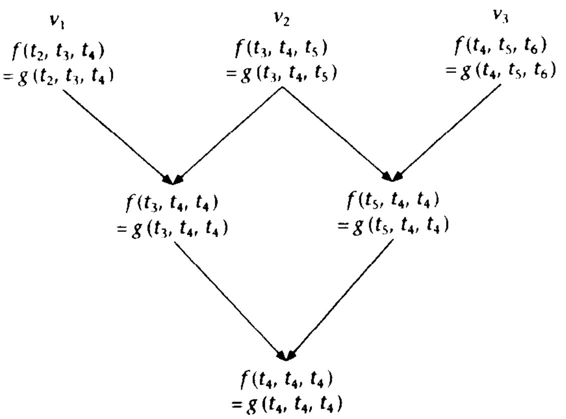


图19-10 在节点t_4处求f和g

结论是用这种方式构造多项式曲线段正好给了我们B样条曲线的含义。事实上可以将它作为定义。

B样条曲线的定义。我们假设有节点值序列可满足式（19-80）。在每个区间$[t_i, t_{i+1}]$上，有一个k阶参数化曲线$F(t)$，由对应的B样条控制点$v_{i-k}, v_{i-k+1}, \cdots, v_i$所定义（对于三次B样条曲线$k=3$）。下面是对B样条的一个构造性定义：

假设$f(\ )$是k参数的极化形式，对应于曲线段在$[t_i, t_{i+1}]$的部分。那么，

（1）控制点由$v_j=f(t_{j+1}, \cdots, t_{j+k})$定义，$j=i-k, i-k+1, \cdots, i$。

（2）对应于这个曲线段的k阶Bézier曲线有控制点。 409

$$p_j = f(\underbrace{t_i,t_i,\cdots,t_i}, t_{i+1},t_{i+1},\cdots,t_{i+1})$$
$$= f(t_i^{[k-j]}, t_{i+1}^{[j]}) \qquad (19\text{-}84)$$
$$j=0,1,\cdots,k$$

（3）曲线在$t\in[t_i, t_{i+1}]$处的计算由$F(t)=f(t, t, \cdots, t)$给出。

实例及Bézier点的关系

现在考虑二次B样条曲线（$k=2$）的情况，并限制在第i个区间$t\in[t_i, t_{i+1}]$。那么对应于这个曲线段的二次Bézier曲线有控制点：

$$\begin{aligned} p_0&=f(t_i, t_i)\\ p_1&=f(t_i, t_{i+1})\\ p_2&=f(t_{i+1}, t_{i+1}) \end{aligned} \qquad (19\text{-}85)$$

B样条控制点是

$$\begin{aligned} v_{i-2}&=f(t_{i-1}, t_i)\\ v_{i-1}&=f(t_i, t_{i+1})\\ v_i&=f(t_{i+1}, t_{i+2}) \end{aligned} \qquad (19\text{-}86)$$

我们有插值：

$$\begin{aligned} f(t_i,t_i)&=\left(\frac{t_{i+1}-t_i}{t_{i+1}-t_{i-1}}\right)v_{i-2}+\left(\frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}\right)v_{i-1}\\ f(t_{i+1},t_{i+1})&=\left(\frac{t_{i+2}-t_{i+1}}{t_{i+2}-t_i}\right)v_{i-1}+\left(\frac{t_{i+1}-t_i}{t_{i+2}-t_i}\right)v_i \end{aligned} \qquad (19\text{-}87)$$

从这能看出（如图19-11）：

- p_0是一个在v_{i-2}和v_{i-1}之间的插值；
- $p_1=v_{i-1}$；
- p_2是在v_{i-1}和v_i之间的插值。 410

我们可以对三次的情形即$k=3$给出一个相似的构造。再一次我们考虑第j个节点区间$[t_j, t_{j+1}]$，以及关联的B样条控制点$v_{j-3}, v_{j-2}, v_{j-1}, v_j$。

所要求的Bézier点是：

$$f(t_j,t_j,t_j), f(t_j,t_j,t_{j+1}), f(t_j,t_{j+1},t_{j+1}), f(t_{j+1},t_{j+1},t_{j+1})$$

构造如图19-12所示。

这个图说明在Bézier点的计算中涉及到两个三角形。注意，所有的四个 Bézier 点都包含在这个表中了。p_1和p_2是对相同的两个B样条控制点的线性插值（但是所用的比率不同），而p_0和p_3是基于B样条控制点参数值的二次插值。同时，p_0和p_3对应于后继曲线的连接点，下一个曲线的p_0等于前一个曲线的p_3。从表中应该能清楚地看出连接点是C^2连续的。图19-13中的二次B样条曲线的渲染是通过将图中 B 样条控制点转换成Bézier控制点完成的。注意曲线是如何位于四个 B 样条控制点的连续序列所构成的凸包内的。

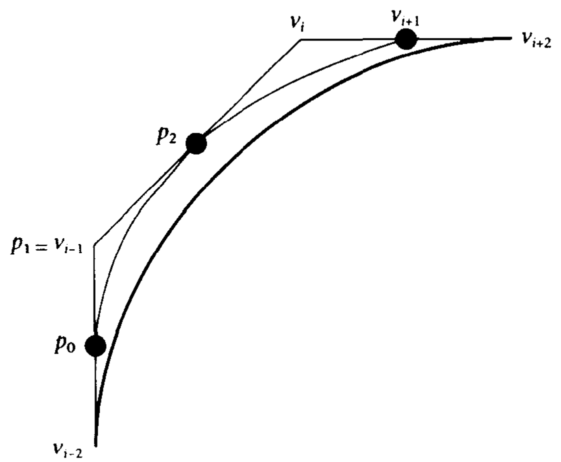


图19-11 二次情形下的B样条点和Bézier点

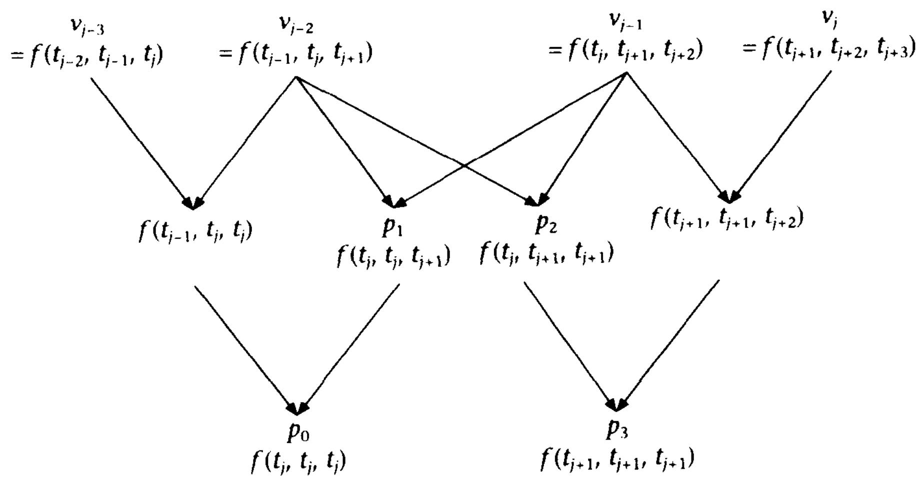


图19-12 从B样条点构造Bézier点

计算(de Boor算法)

411 现在考虑B样条对应于参数值 $t \in [t_i, t_{i+1}]$的点的计算问题。我们知道可以用$f(t, t, \cdots, t)$计算。

从图19-14我们可以建立下列递归公式，称为计算k阶 B 样条曲线的 de Boor 算法:

$$P_j^0(t) = v_j, j = i-k, i-k+1, \cdots, i$$

$$P_j^r(t) = \left(\frac{t_{k+1+j} - t}{t_{k+1+j} - t_{r+j}}\right) P_j^{r-1}(t) + \left(\frac{t - t_{r+j}}{t_{k+1+j} - t_{r+j}}\right) P_{j+1}^{r-1}(t) \quad (19\text{-}88)$$

$$r = 1, 2, \cdots, k \text{ 且 } j = i-k, i-k+1, \cdots, i-r$$

那么曲线上所要求的点是 $P_{i-k}^k(t)$。

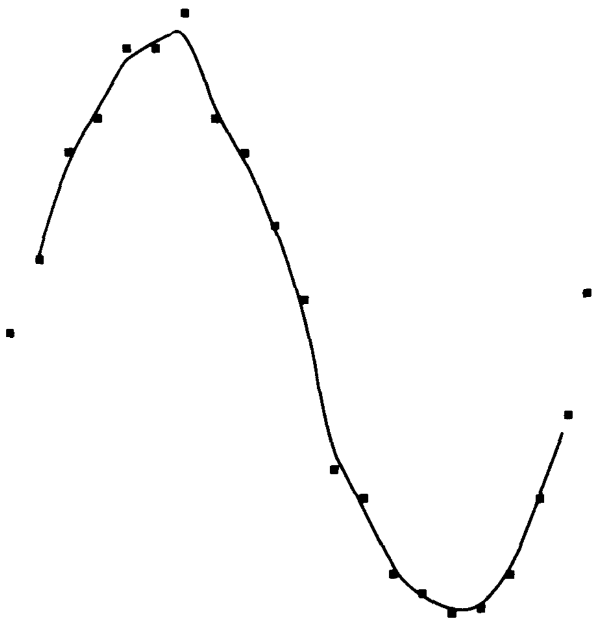


图19-13 从Bézier曲线渲染B样条曲线

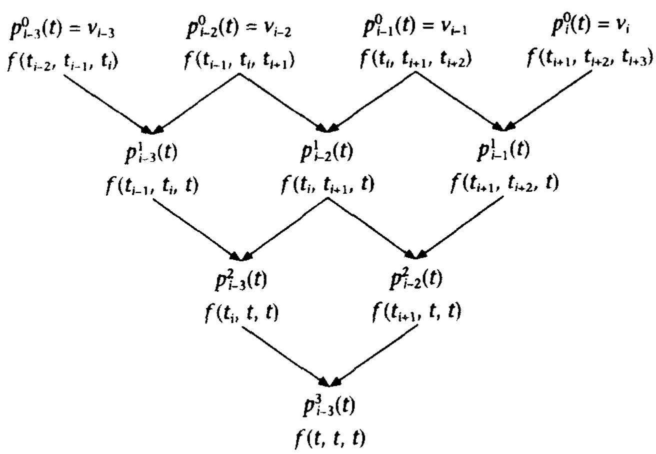


图19-14 计算B样条曲线的de Boor算法

节点序列和控制点的标记

假设我们有三次B样条的控制点v_0，v_1，…，v_n。该序列对应的节点矢量应该是什么？设节

点用t_1、t_2、…标记。

假设f是极化形式，因此我们知道在三次的情况下$v_0=f(t_1, t_2, t_3)$且$v_n=f(t_{n+1}, t_{n+2}, t_{n+3})$。在曲线的开始处，节点$t_1$、…、$t_6$允许曲线计算在参数范围 [$t_3$, t_4]内，并使用控制点v_0、…、v_4。同样地对曲线终点，最后的参数范围是 [t_n, t_{n+1}]。

412 因此，对于三次情形有下列各项结论：

- 对于控制点 v_0、v_1、…、v_n所要求的节点序列是 t_1、t_2、…、t_{n+3}。
- 曲线定义在范围 $t \in [t_3, t_{n+1}]$内。
- 总共会有$n-2$个曲线段，因为每个区间 [t_i, t_{i+1}]，i=3、4、…、n定义一个曲线段。

在第一个区间[t_3, t_4]中构造三角形队列对应于曲线计算。现在假设前三个节点是相等的，即$t_1=t_2=t_3=t_0$。这样很容易看出$t=t_0$对曲线计算的结果是v_0。对于最后三个节点值相等的情况也有相似的结论：

如果最初的三个节点值是相等的，那么B样条曲线将会在第一个控制点处开始。同样地它将会在最后控制点处结束，如果最后三个节点是相等的。

这是更一般的结果的一个特例：

每次一个节点值被重复，连续性的阶在对应于那个节点的连接点处将损失1。一般来讲，如果B样条基于k阶多项式，而且一个特别的节点值重复次数为m，那么在这个结合点上连续性的阶是$k-m$。

节点插入

插入新的节点到序列中是维持同一条B样条曲线的一个基本操作。两个应用是：

- 渲染。
- 413 为曲线形状增加更大的灵活性。

对此有许多算法（Goldman，1990），我们在这一小节中只介绍 Boehm节点插入算法，而且只讨论三次的情况。其思想是在一个区间[t_i, t_{i+1}]中，插入一个新的节点$\hat{t}$。因此在这个局部区域中节点序列将是：

$$t_{i-2},t_{i-1},t_i,\hat{t},t_{i+1},t_{i+2},t_{i+3} \tag{19-89}$$

其对应的控制点是：

$$\begin{aligned} f(t_{i-2},t_{i-1},t_i) &= v_{i-3} \\ f(t_{i-1},t_i,\hat{t}) &= w_1 \\ f(t_i,\hat{t},t_{i+1}) &= w_2 \\ f(\hat{t},t_{i+1},t_{i+2}) &= w_3 \\ f(t_{i+1},t_{i+2},t_{i+3}) &= v_i \end{aligned} \tag{19-90}$$

设旧的控制点是 v_0、v_1、…、v_n，在区间[t_i, t_{i+1}]中插入一个节点，并设完整的新的控制点集合是w_0、w_1、…、w_n、w_{n+1}。

那么可以看到有下列关系存在：

$$\begin{aligned} & w_j = v_j, j = 0,1,\cdots,i-3 \\ & w_j = f(t_{j+1},t_{j+2},\hat{t}), j = i-2,i-1,i \\ & w_j = v_{j-1}, j = i+1,i+2,\cdots,n+1 \end{aligned} \tag{19-91}$$

还需要求出显式求解$f(t_{j+1}, t_{j+2}, \hat{t})$，$j=i-2, i-1, i$的公式。这很容易通过使用极化形式的多仿射性质来完成。由$f(t_{j+1}, t_{j+2}, \hat{t})$所表示的控制点落在$f(t_j, t_{j+1}, t_{j+2})$和$f(t_{j+1}, t_{j+2}, t_{j+3})$之间。因而我们可以使用通常的插值结果，在$\hat{t} \in [t_j, t_{j+3}]$上插值有：

$$f(t_{j+1},t_{j+2},\hat{t})=\left(\frac{t_{j+3}-\hat{t}}{t_{j+3}-t_j}\right)f(t_j,t_{j+1},t_{j+2})+\left(\frac{\hat{t}-t_j}{t_{j+3}-t_j}\right)f(t_{j+1},t_{j+2},t_{j+3}) \tag{19-92}$$

所以有

$$w_j=\left(\frac{t_{j+3}-\hat{t}}{t_{j+3}-t_j}\right)v_{j-1}+\left(\frac{\hat{t}-t_j}{t_{j+3}-t_j}\right)v_j \qquad j=i-2,i-1,i \tag{19-93}$$

这称为Boehm节点插入公式。为了插入多个节点，这个过程需要为每一个节点重复一次。节点插入对于渲染是有用的，因为可以证明当越来越多的节点被插入节点序列中时，在两端点之间控制点的序列越来越接近B样条曲线。

多节点插入——Oslo算法

在Boehm节点插入算法中，我们一次在一个区间中插入一个新的节点，得到k个新的控制点（这里k是参数化多项式的阶），虽然与先前的情形相比只有一个额外的控制点。Oslo算法同时在一个区间中插入多个节点。我们在这里给出Oslo算法的一个简单版本，是由Goldman（1990）给出的，基于开花。 414

我们继续讨论$k=3$的情况，并且考虑在区间$[t_i, t_{i+1}]$中插入三个节点，那么靠近这个区间的新节点序列是：

$$t_{i-2},t_{i-1},t_i,\hat{t}_1,\hat{t}_2,\hat{t}_3,t_{i+1},t_{i+2},t_{i+3} \tag{19-94}$$

因此控制点是：

$$\begin{aligned}
&v_{i-1}=f(t_{i-2},t_{i-1},t_i)\ \text{（最初的控制点）}\\
&\text{（新的控制点）}\\
&w_1=f(t_{i-1},t_i,\hat{t}_1)\\
&w_2=f(t_i,\hat{t}_1,\hat{t}_2)\\
&w_3=f(\hat{t}_1,\hat{t}_2,\hat{t}_3)\\
&w_4=f(\hat{t}_2,\hat{t}_3,t_{i+1})\\
&w_5=f(\hat{t}_3,t_{i+1},t_{i+2})\\
&v_i=f(t_{i+1},t_{i+2},t_{i+3})\ \text{（最初的控制点）}
\end{aligned} \tag{19-95}$$

那么问题变成该如何安排一个适当的计算来获得这些新的控制点。这可以通过构造两个de Casteljau三角形来完成。第一个是插入节点，以$\hat{t}_1,\hat{t}_2,\hat{t}_3$的顺序（如图19-15）一行一行地进行，第二个以相反的顺序（如图19-16）进行。

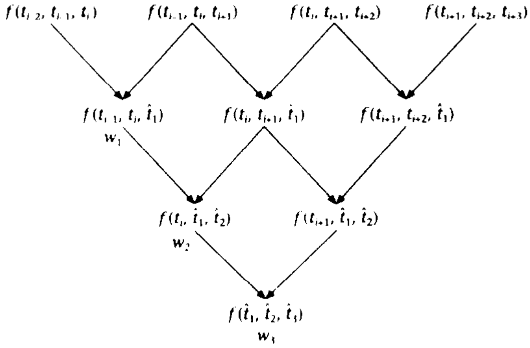


图19-15　按升序插入节点

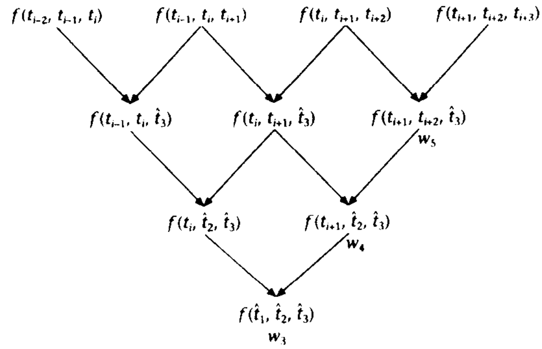


图19-16　按降序插入节点

19.9　B样条基函数

在前面我们看到了Bézier曲线如何使用 Bernstein 基表示。我们可以使用熟悉的方法来表示B样条曲线，即将k阶B样条曲线表示成B样条基函数相似权值的总和：

415

$$F(t)=\sum_{i=0}^{n}N_{k,i}(t)v_i \tag{19-96}$$

这里基函数由递归公式定义：

$$N_{0,i}(t)=\begin{cases}1 & t\in[t_i,t_{i+1}]\\ 0 & \text{其他}\end{cases}$$

$$N_{r,i}(t)=\left(\frac{t-t_i}{t_{i+r}-t_i}\right)N_{r-1,i}(t)+\left(\frac{t_{i+r+1}-t}{t_{i+r+1}-t_{i+1}}\right)N_{r-1,i+1}(t),t\in[t_i,t_{i+r+1}] \tag{19-97}$$

$N_{r,i}(t)$ 是r阶多项式，在$[t_i,\ t_{i+r+1}]$上有局部支撑，即只是在这个范围上是非零的。它由$r+1$个r阶多项式曲线段所组成，在连接处有C^{r-1}连续性。Seidel(1989)给出了如何从B样条表示中导出开花方法。这里我们说明如何从开花表示出发，最后以基函数（式（19-97））结束，假设用式（19-96）的形式表达 B 样条曲线的需求。

首先考虑情况$k=1$，线段包括点v_i，设开花始终是f。这些线段中的第一个是 $t\in[t_i,\ t_{i+1}]$，有$v_{i-1}=f(t_i)$和 $v_i=f(t_{i+1})$（这些是单参数极化形式的一阶多项式）。那么，

$$F(t)=\left(\frac{t_{i+1}-t}{t_{i+1}-t_i}\right)v_{i-1}+\left(\frac{t-t_i}{t_{i+1}-t_i}\right)v_i,\quad t\in[t_i,t_{i+1}] \tag{19-98}$$

然而，v_i也对下一线段有贡献，$t\in[t_{i+1},\ t_{i+2}]$，$v_i=f(t_{i+1})$和 $v_{i+1}=f(t_{i+1})$。因此，

416

$$F(t)=\left(\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}\right)v_i+\left(\frac{t-t_{i+1}}{t_{i+2}-t_{i+1}}\right)v_{i+1},\quad t\in[t_{i+1},t_{i+2}] \tag{19-99}$$

为了要将$F(t)$写成式（19-96）的形式，每个v_i必须与单一系数$N_{1,i}(t)$相关联。将式（19-98）和式（19-99）中的v_i系数设为与式（19-96）中的相等，我们有：

$$N_{1,i}(t)=\left(\frac{t-t_i}{t_{i+1}-t_i}\right)N_{0,i}(t)+\left(\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}\right)N_{0,i+1}(t),\quad t\in[t_i,t_{i+2}] \tag{19-100}$$

它是式（19-97）在r=1的一个特例。

让我们稍微深入一步，考虑r=2的情况。重新考虑v_i对之有贡献的三个线段：

（1）线段$t \in [t_i, t_{i+1}]$，包括$v_{i-2}= f(t_{i-1}, t_i)$，$v_{i-1}=f(t_i, t_{i+1})$和$v_i=f(t_{i+1}, t_{i+2})$。通过使用de Casteljau三角形（见图19-17），很容易看到在$F(t)$中v_i的系数是：

$$\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\left(\frac{t-t_i}{t_{i+1}-t_i}\right) \tag{19-101}$$

（2）线段$t \in [t_{i+1}, t_{i+2}]$，包括$v_{i-1}= f(t_i, t_{i+1})$，$v_i=f(t_{i+1}, t_{i+2})$和$v_{i+1}= f(t_{i+2}, t_{i+3})$。这里$v_i$的系数是（见图19-18）：

$$\left(\frac{t-t_i}{t_{i+2}-t_i}\right)\left(\frac{t_{i+2}-t}{t_{i+2}-t_{i+1}}\right)+\left(\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}\right)\left(\frac{t-t_{i+1}}{t_{i+2}-t_{i+1}}\right) \tag{19-102}$$

（3）线段$t \in [t_{i+2}, t_{i+3}]$，包括$v_i=f(t_{i+1}, t_{i+2})$，$v_{i+1}= f(t_{i+2}, t_{i+3})$和$v_{i+2}=f(t_{i+3}, t_{i+4})$。$v_i$的系数是： 417

$$\left(\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}\right)\left(\frac{t_{i+3}-t}{t_{i+3}-t_{i+2}}\right) \tag{19-103}$$

现在使用（1）、（2）和（3），式（19-96）中v_i的系数在r=2时一定是：

$$N_{2,i}(t)=\left(\frac{t-t_i}{t_{i+2}-t_i}\right)N_{1,i}(t)+\left(\frac{t_{i+3}-t}{t_{i+3}-t_{i+1}}\right)N_{1,i+1}(t), \quad t\in[t_i,t_{i+3}] \tag{19-104}$$

这是式（19-97）对于r=2的一个特例。一般结果可以通过对r归纳得出。

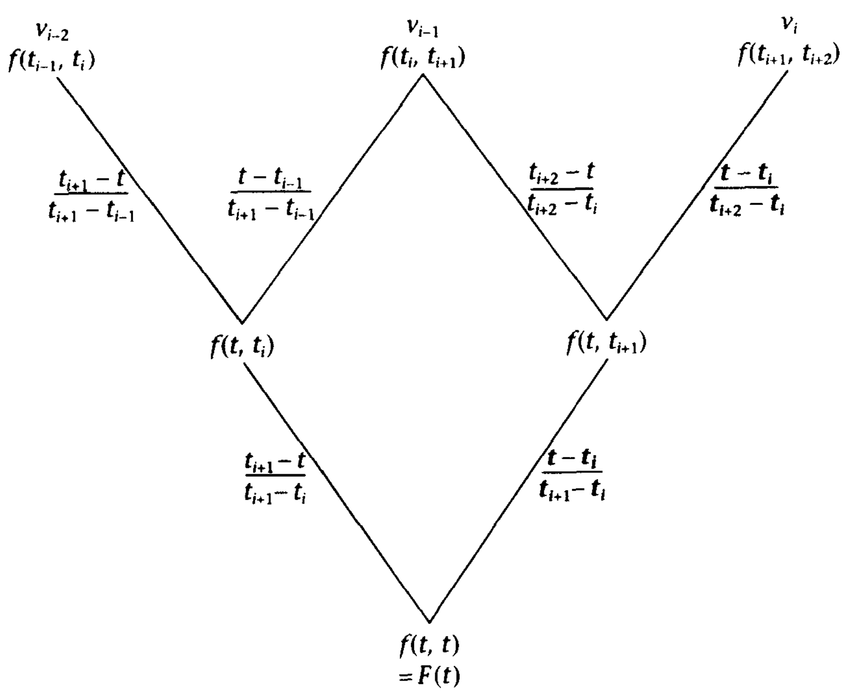


图19-17 v_i的系数是沿粗体字路径获得的

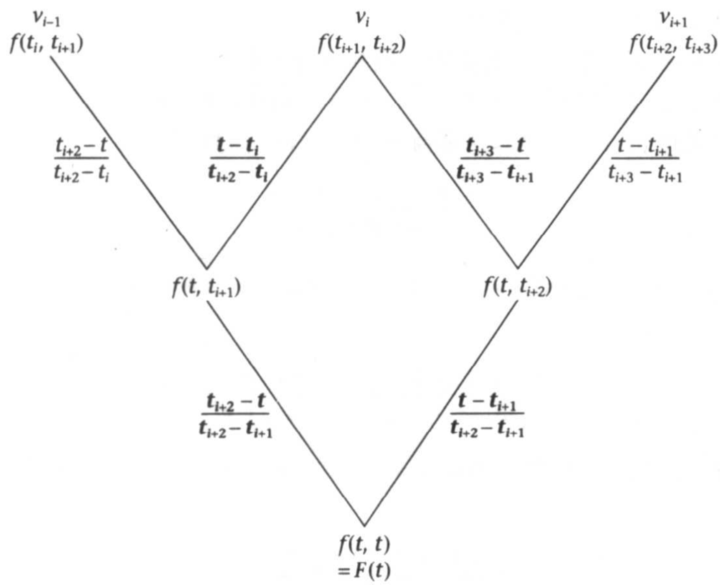


图19-18 v_i的系数是沿粗体字路径获得的

19.10 对曲面的介绍

在下几个小节中我们将说明曲线理论如何能非常容易地应用到曲面的构造，如张量积表面、直纹面或矩形面片（这些术语是可互换的）。曲线理论可以非常直接地应用到这种曲面，基本上没有什么新的思想需要介绍。而曲线是通过控制点序列和节点序列（或参数区间）来描述的，因此在 3D 中有一个控制点矩形队列，并有两个节点序列来定义参数区间。

一个曲面是由下列形式的函数参数化描述的：

$$F(t,u)=(X(t,u),Y(t,u),Z(t,u)),\quad t,u\in[0,1] \tag{19-105}$$

418 （参数范围是任意的，为方便起见，选择[0，1]。）

很容易看出为什么它定义的是一个曲面。对任何固定的t，当u在0和1之间变动的时候，沿着路径$(X(t,u), Y(t,u), Z(t,u))$将绘制出一条曲线。取这条曲线上的任意一点，现在允许t改变，很显然点还将在某条曲线上运动。通过这种方法我们可以看到这将是一个曲线的曲线，也就是说定义的是一个曲面。

这由图19-19说明，图中清楚地标明了曲面上u和t的方向。考虑这样一条曲线A，它是固定u值变化t得到的。那么取曲线上的任意一点，允许u改变，我们将又得到另外一条曲线，比如B。整个曲面就是这样定义的。

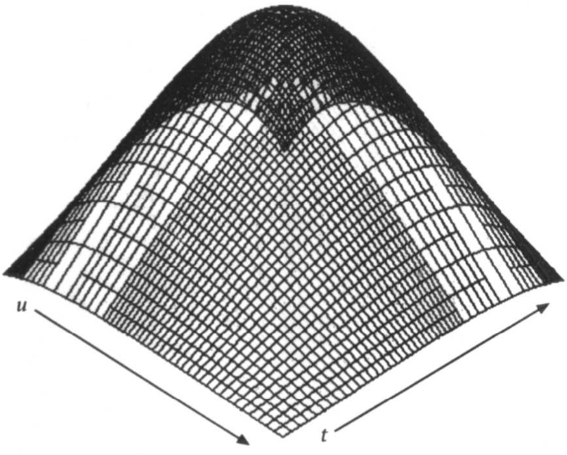


图19-19 参数化曲面：这是一个Bézier曲面，其控制点位于高斯曲面上

在计算机辅助几何设计中，形式$(X(t,$

$u)$、$Y(t、u)$、$Z(t、u)$)称为以u和t为参数的多项式表示。精确的表示形式确定了曲面的类型——我们专注于讨论Bézier和B样条形式，如曲线一样。

19.11 参数化曲面

Bernstein基表示

假如有一个$(m+1)\times(n+1)$的 Bézier 控制点队列如下所示：

$$\begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ p_{10} & p_{11} & \cdots & p_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m0} & p_{m1} & \cdots & p_{mn} \end{bmatrix} \tag{19-106}$$

每一行或列都定义了一条Bézier曲线。我们能通过所谓的Bézier曲线的张量积来形成曲线的曲线，即一个曲面：

$$F(t,u)=\sum_{i=0}^{m}\sum_{j=0}^{n}B_{m,i}(t)p_{ij}B_{n,j}(u) \tag{19-107}$$ 419

$$t,u\in[0,1]$$

这里基函数与式（19-33）中的相同。

我们将式（19-107）重新整理得：

$$F(t,u)=\sum_{i=0}^{m}B_{m,i}(t)\left(\sum_{j=0}^{n}p_{ij}B_{n,j}(u)\right) \tag{19-108}$$

对固定的t和i，在括号中的项为第 i行控制点所对应的Bézier曲线。对任何固定的u，括号中的表达式求那条曲线上的一个点。经过i的每个值，另一条Bézier曲线就生成了。因此Bézier 曲面可以被看成Bézier曲线的Bézier曲线。

渲染Bézier曲线我们可以应用 de Casteljau 算法来拆分每一行，然后进而拆分所产生的每一列。因此初始$m\times n$控制点队列变成四个这种队列的集合。继续这样拆分直到队列逼近共平面的一组控制点，这组控制点可以用四边形逼近。因此，Bézier 面片是通过一组四边平面多边形来逼近的。

```
typedef struct{
  float x,y,z;
}Point3D;

typedef Point3D ControlPointArray[4][4];

void Bezier3D(ControlPointArray p){
  ControlPointArray q,r,s,t;
  if(Coplanar(p)) RenderPolygon(p[0][0],p[3][0],p[3][3],p[0][3]);
  else{
    /*split p into q,r,s,t*/
    Split3D(p,q,r,s,t);
    Bezier3D(q);
    Bezier3D(r);
    Bezier3D(s);
    Bezier3D(t);
  }
}
```

为了要实现`Split3D`，可以对每一行然后对所得的每一列使用de Casteljau 算法，每一次在参数值1/2处拆分。为了测试共平面，可以使用这样的结论，即点(X, Y, Z)到平面$ax+by+cz-d=0$的垂直距离(D)由下式给出：

420
$$D^2=\frac{(ax+by+cz-d)^2}{a^2+b^2+c^2} \tag{19-109}$$

这是代价很大的计算，因为对于双三次 Bézier曲面，它必须对16个点中的13个点各执行一遍，剩余3个点用来计算平面方程。为此，人们提出了许多降低计算代价的逼近算法（Rappaport，1991b）。

矩形Bézier面片的开花

Bézier曲面的极化形式的思想直接来自于Bézier曲线，虽然记号不可避免地更加散乱。作为一个例子，我们考虑如下的双二次Bézier曲面：

$$\begin{aligned}F(t,u)&=\sum_{i=0}^{2}B_{2,i}(t)\left(\sum_{j=0}^{2}p_{ij}B_{2,j}(u)\right)\\&=\sum_{i=0}^{2}B_{2,i}(t)(p_{i0}(1-u)^2+2p_{i1}(1-u)u+p_{i2}u^2)\\&=(1-t^2)(p_{00}(1-u)^2+2p_{01}(1-u)u+p_{02}u^2)\\&\quad+2(1-t)t(p_{10}(1-u)^2+2p_{11}(1-u)u+p_{12}u^2)\\&\quad+t^2(p_{20}(1-u)^2+2p_{21}(1-u)u+p_{22}u^2)\end{aligned} \tag{19-110}$$

这种表达式显然是t和u的双二次函数，其一般形式是：

$$\begin{aligned}F(t,u)=&a_0(b_0+b_1u+b_2u^2)+a_1t(c_0+c_1u+c_2u^2)\\&+a_2t^2(d_0+d_1u+d_2u^2)\end{aligned} \tag{19-111}$$

假设我们首先按t开花这个表达式，将u当作常数，有：

$$a_0(b_0+b_1u+b_2u^2)+a_1\left(\frac{t_1+t_2}{2}\right)(c_0+c_1u+c_2u^2)+a_2t_1t_2(d_0+d_1u+d_2u^2) \tag{19-112}$$

现在将这个表达式按u开花，有：

$$\begin{aligned}&a_0\left(b_0+b_1\left(\frac{u_1+u_2}{2}\right)+b_2u_1u_2\right)\\&+a_1\left(\frac{t_1+t_2}{2}\right)\left(c_0+c_1\left(\frac{u_1+u_2}{2}\right)+c_2u_1u_2\right)\\&+a_2t_1t_2\left(d_0+d_1\left(\frac{u_1+u_2}{2}\right)+d_2u_1u_2\right)\end{aligned} \tag{19-113}$$

421 显然这不是一个非常优美的表达式，虽然容易导出。关键在于它说明了开花是可以形成的，而且可以写成$f(t_1, t_2; u_1, u_2)$的形式。分号指示多项式首先按一个参数开花，然后对所得结果按另一个参数开花。

至此Bézier控制点和开花之间的关系可以直接从曲线情形得出。通常，对于 $m\times n$ Bézier曲面，$t\in[a, b]$和$u\in[c, d]$：

$$\begin{aligned} p_{ij} &= f(a,a,\cdots,a,b,b,\cdots,b;c,c,\cdots,c,d,d,\cdots,d) \\ &= f(a^{[m-i]}b^{[i]};c^{[n-j]}d^{[j]}) \end{aligned} \tag{19-114}$$

（其中 $b^{[i]}$表示b、b、…、b——即长度为i的b值序列。）

B样条曲面

理论再一次直接来自于曲线。考虑双三次曲面的情况。我们有$m \times n$的B样条的控制点v_{ij}的队列。我们有两个节点序列，一个为行，另一个为列：

- t_1，t_2，…，t_{m+3}是对每一列；
- u_1，u_2，…，u_{n+3}是对每一行。

所有来自曲线的定理都可直接应用。举例来说：

（1）对任何行或列的控制点间连接以及开花都如前一样。因此使用先前一节中的记号：

$$\begin{aligned} v_{ij} &= f(t_{i+1},t_{i+2},t_{i+3};u_{j+1},u_{j+2},u_{j+3}) \\ i &= 0,1,2,\cdots,m \\ j &= 0,1,2,\cdots,n \end{aligned} \tag{19-115}$$

（2）从式（19-115）我们可以单独在行和列上使用插值，以便导出任意单个面片的Bézier控制点。举例来说，从式（19-114）我们知道由范围$[t_i, t_{i+1}]$和$[u_j, u_{j+1}]$所定义的面片的控制点将有如下形式的Bézier 控制点：

$$\begin{aligned} p_{ab} &= f(t_i^{[3-a]}t_{i+1}^{[a]};u_j^{[b]}u_{j+1}^{[3-b]}) \\ a,b &= 0,1,2,3 \end{aligned} \tag{19-116}$$

（3）节点插入或许是最容易的渲染曲面的方法，可以通过对每行插入节点然后再对每列插入节点，其模式与前面所介绍的一样。

19.12 三角形Bézier面片

前一小节已经讨论了矩形面片，即参数域是在某个区间 $[a, b] \times [c, d]$上的。而且，产生曲面的多项式的阶有形式$m \times n$，这里第一个参数是阶m，第二个参数是阶n。这里我们考虑另一种形式，在很多时候都简单得多，这里参数域是一个三角形，对应多项式的总的阶是n。它们被称为三角形面片。 422

图19-20给出了R^2中顶点由**r**、**s** 和**t**定义的三角形参数区域。任何参数值$\mathbf{u} \in R^2$可以被表示成顶点的一个重心组合：

$$\begin{aligned} \mathbf{u} &= \alpha\mathbf{r} + \beta\mathbf{s} + \gamma\mathbf{t} \\ \alpha+\beta+\gamma &= 1 \\ \alpha,\beta,\gamma &\geqslant 0 \end{aligned} \tag{19-117}$$

设$F(\mathbf{u})$是$R^2 \to R^3$的一个n阶多项式表达式，用这些重心坐标表达且总阶数为n。那么

$$F(\mathbf{u}) = \sum_{i+j+k=n} a_{ijk}\alpha^i\beta^j\gamma^k \tag{19-118}$$

这里 $a_{ijk} \in R^3$（已知（α, β, γ）为**u**的一个重心坐标）。

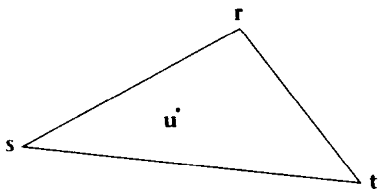


图19-20 三角形参数域：任何参数值u都可以表示为重心坐标（α, β, γ）

现在可以用如参数域多项式相同的方法构造出多项式的开花。举例来说，假设有n=3，有

$$F(\mathbf{u}) = \alpha^3 + 6\alpha^2\beta + \beta\gamma^2 + \alpha\beta\gamma \tag{19-119}$$

那么

$$\begin{aligned} f(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3) = \alpha_1\alpha_2\alpha_3 &+ 6\left(\frac{\alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_2\alpha_3}{3}\right)\left(\frac{\beta_1 + \beta_2 + \beta_3}{3}\right) \\ &+ \left(\frac{\beta_1 + \beta_2 + \beta_3}{3}\right)\left(\frac{\gamma_1\gamma_2 + \gamma_1\gamma_3 + \gamma_2\gamma_3}{3}\right) \\ &+ \left(\frac{\alpha_1 + \alpha_2 + \alpha_3}{3}\right)\left(\frac{\beta_1 + \beta_2 + \beta_3}{3}\right)\left(\frac{\gamma_1 + \gamma_2 + \gamma_3}{3}\right) \end{aligned} \tag{19-120}$$

显然式（19-120）有开花必须的所有性质——它是多仿射的、对称的，而且它的对角线等于对应的多项式（式(19-119)）。

通常对应于任何总阶数为n的多项式函数$F(\mathbf{u})$，有一个惟一对应的开花$f(\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n)$，它是多仿射的、对称的，且其对角线$f(\mathbf{u}, \mathbf{u}, \cdots, \mathbf{u})=F(\mathbf{u})$。

这可以用来构造 Bézier 三角形面片。考虑：

$$F(\mathbf{u}) = f(\mathbf{u}, \mathbf{u}, \cdots, \mathbf{u}) \tag{19-121}$$

从式（19-117），我们可以使用多仿射性质得：

423 $$F(\mathbf{u}) = \alpha f(\mathbf{r}, \mathbf{u}, \cdots, \mathbf{u}) + \beta f(\mathbf{s}, \mathbf{u}, \cdots, \mathbf{u}) + \gamma f(\mathbf{t}, \mathbf{u}, \cdots, \mathbf{u}) \tag{19-122}$$

不断用这种方式展开，我们得到：

$$F(\mathbf{u}) = \sum_{i+j+k=n} \left(\frac{n!}{i!\,j!\,k!}\right) \alpha^i \beta^j \gamma^k f(\mathbf{r}^{[i]}, \mathbf{s}^{[j]}, \mathbf{t}^{[k]}) \tag{19-123}$$

如果把开花值与3D空间中的点联系在一起：

$$p_{ijk} = f(\mathbf{r}^{[i]}, \mathbf{s}^{[j]}, \mathbf{t}^{[k]}) \tag{19-124}$$

我们得到n 阶 Bézier 三角形面片。

考虑n =2的情况，那么三角形的排列为：

$$\begin{matrix} & & p_{200} & & \\ & p_{110} & & p_{101} & \\ p_{020} & & p_{011} & & p_{002} \end{matrix} \tag{19-125}$$

对应于开花的排列：

$$\begin{matrix} & & f(\mathbf{r},\mathbf{r}) & & \\ & f(\mathbf{r},\mathbf{s}) & & f(\mathbf{r},\mathbf{t}) & \\ f(\mathbf{s},\mathbf{s}) & & f(\mathbf{s},\mathbf{t}) & & f(\mathbf{t},\mathbf{t}) \end{matrix} \tag{19-126}$$

同样地对于 n=3的情况：

$$
\begin{array}{c}
p_{300} \\
p_{210} \quad p_{201} \\
p_{120} \quad p_{111} \quad p_{102} \\
p_{030} \quad p_{021} \quad p_{012} \quad p_{003}
\end{array}
\tag{19-127}
$$

对应于：

$$
\begin{array}{c}
f(\mathbf{r},\mathbf{r},\mathbf{r}) \\
f(\mathbf{r},\mathbf{r},\mathbf{s}) \quad f(\mathbf{r},\mathbf{r},\mathbf{t}) \\
f(\mathbf{r},\mathbf{s},\mathbf{s}) \quad f(\mathbf{r},\mathbf{s},\mathbf{t}) \quad f(\mathbf{r},\mathbf{t},\mathbf{t}) \\
f(\mathbf{s},\mathbf{s},\mathbf{s}) \quad f(\mathbf{s},\mathbf{s},\mathbf{t}) \quad f(\mathbf{s},\mathbf{t},\mathbf{t}) \quad f(\mathbf{t},\mathbf{t},\mathbf{t})
\end{array}
\tag{19-128}
$$

使用多仿射性质可以立刻使用这个排列来求出曲面上的任何点 $f(\mathbf{u}, \mathbf{u}, \mathbf{u})$。这由图19-21说明，这里我们在每个内部三角形上插值来获得新的三角形，并重复直到得到了$f(\mathbf{u}, \mathbf{u}, \mathbf{u})$。每个插值都有式（19-122）的形式。首先，对最初控制点插值来产生标记为1的内部点。然后，对这些进行插值产生标记为 2 的点。最后所要的点标记为3。

注意在图19-21中的插值产生三个三角形，每个都有式（19-128）这样的相同形式。（考虑四行的三角形，其顶点是 $f(\mathbf{u}, \mathbf{u}, \mathbf{u})$，它的最后一行是最初三角形的最后一行）。这说明三角形面片可以被分解为 de Casteljau 细分，由此我们可以得到如上的简单递归渲染模式。

424

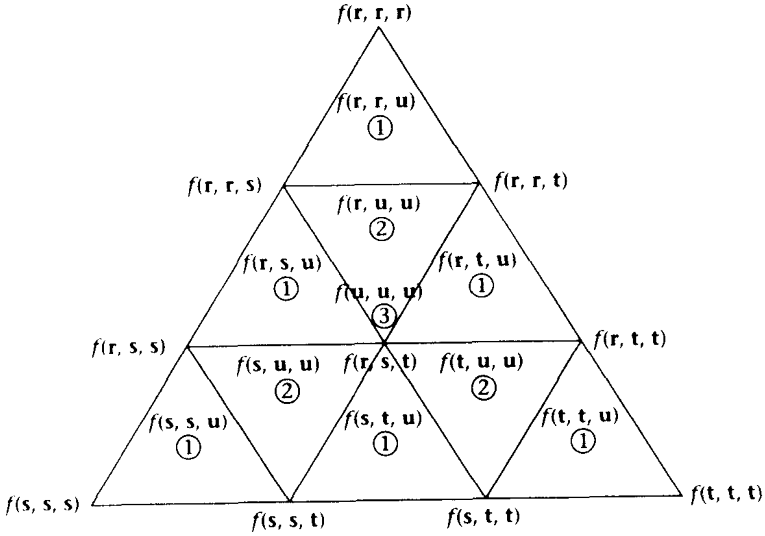


图19-21 三角形面片的插值模式

19.13 三次B样条插值

曲线插值问题

我们在整个这一章中都专注于讨论曲线和曲面的设计问题，这是基于一组控制点和曲线或曲面之间关系的曲线和曲面构造。在这一小节中我们考虑相反的问题：假设给定一条曲线上的一些点，我们要求出确定这条曲线的控制点。当给定n个不同数据点序列 p_1，p_2，…，p_n，

对三次B样条曲线 F，

$$F(t_i) = p_i \quad i = 1,2,\cdots,n \tag{19-129}$$

这里t_i包含一个给定序列的节点值，以升序排列。我们要求出构造这样一条B样条曲线的控制点v_j。假设对应于F的开花是3参数的极化形式f（u_1，u_2，u_3）。

三次B样条曲线的解

为了解决这个问题，我们首先考察所需要的节点序列。因为我们知道曲线一定开始于p_1点，最初的三个节点值应该是 t_1，这样给定：

$$p_1 = f(t_1,t_1,t_1) \tag{19-130}$$

同样地在曲线的另一端：

425 $$p_n = f(t_n,t_n,t_n) \tag{19-131}$$

因此节点序列一定有形式：

$$t_1,t_1,t_1,t_2,t_3,t_4,\cdots,t_{n-4},t_{n-3},t_{n-2},t_{n-1},t_n,t_n,t_n \tag{19-132}$$

通过B样条曲线的定义，在通常的节点序列上有

$$t_1,t_2,t_3,t_4,\cdots,t_{n-4},t_{n-3},t_{n-2},t_{n-1},t_n \tag{19-133}$$

我们有：

$$v_i = f(t_{i+1},t_{i+2},t_{i+3}) \quad i = 0,1,2,\cdots,n-3 \tag{19-134}$$

对于这个节点值范围，用于求解p_i的 de Casteljau 三角形如图19-22 所示。

从这一点我们可以导出一组方程

$$p_i = a_{i-3}v_{i-3} + b_{i-2}v_{i-2} + c_{i-1}v_{i-1} \quad i = 3,4,\cdots,n-2 \tag{19-135}$$

这里系数a_{i-3}、b_{i-2}和c_{i-1}是节点值的函数，可以用通常的方式在三角形上求得。当然在这些方程中未知的是控制点v_j。至此我们有$n-4$个方程和$n-2$个未知数。

426 现在考虑节点序列的起点并求解p_2，如图19-23所示。

由此可见：

$$p_2 = a_{-1}v_{-1} + b_0v_0 + c_1v_1 \tag{19-136}$$

节点序列的另一端如图19-24所示。由此又可得：

$$p_{n-1} = a_{n-4}v_{n-4} + b_{n-3}v_{n-3} + c_{n-2}v_{n-2} \tag{19-137}$$

将式（19-135）、式（19-136）和式（19-137）放在一起，我们有：

$$p_i = a_{i-3}v_{i-3} + b_{i-2}v_{i-2} + c_{i-1}v_{i-1} \quad i = 2,4,\cdots,n-1 \tag{19-138}$$

现在我们总共有$n-2$个方程，有n个未知数v_{-1}，v_0，…，v_{n-2}。这留给我们两个自由度，设

$$v_{-1} = q_1 \quad v_{n-2} = q_n \tag{19-139}$$

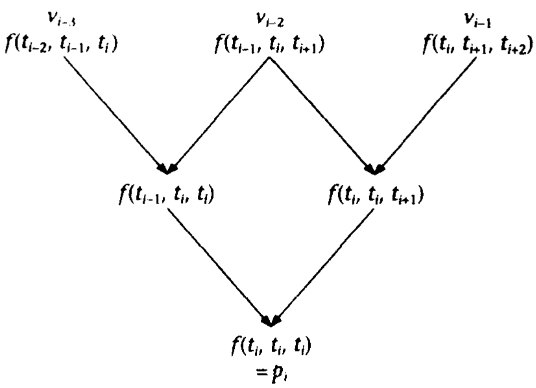


图19-22　求式（19-133）中节点的数据点

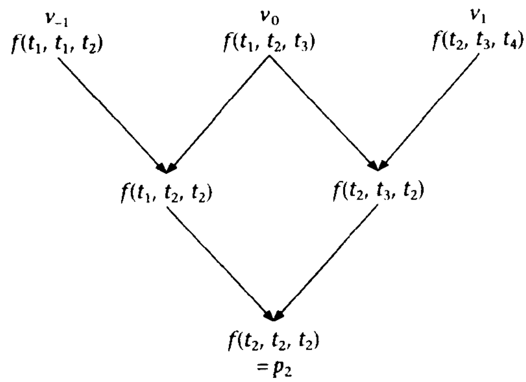


图19-23　求p_2

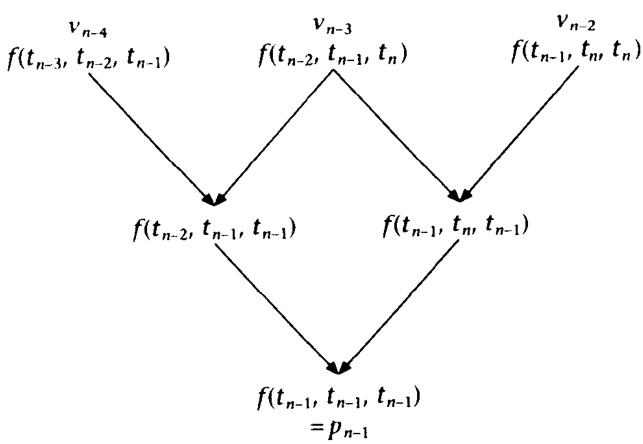


图19-24　求p_{n-1}

是任意的（这些将会影响曲线靠近终点处的形状），这样就得到了与未知数相同数目的方程。

矩阵方程

总的方程组现在可以写成下列形式： 427

$$
\begin{aligned}
v_{-1} &= q_1 \quad（选定的）\\
a_{-1}v_{-1} + b_0 v_0 + c_1 v_1 &= p_2\\
a_0 v_0 + b_1 v_1 + c_2 v_2 &= p_3\\
&\cdots\\
a_{n-4}v_{n-4} + b_{n-3}v_{n-3} + c_{n-2}v_{n-2} &= p_{n-1}\\
v_{n-2} &= q_n \quad（选定的）
\end{aligned}
\tag{19-140}
$$

这可以写成矩阵的形式（如式（19-141）），从中可以看出这是一个三对角阵方程，可以特别简单地求解。

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ a_{-1} & b_0 & c_1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & a_0 & b_1 & c_2 & 0 & \cdots & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & a_{n-4} & b_{n-3} & c_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{-1} \\ v_0 \\ v_1 \\ \cdots \\ v_{n-3} \\ v_{n-2} \end{bmatrix} = \begin{bmatrix} q_1 \\ p_2 \\ p_3 \\ \cdots \\ p_{n-1} \\ q_n \end{bmatrix} \tag{19-141}$$

有一点是非常重要的，那就是v_j和p_i要么是二维中的点、要么是三维中的点。因此有两或三个这样的线性方程组需要求解。

19.14　求解多项式

前向差分

我们在本章最后考虑一个基本问题：多项式的快速求解。这一小节首先考虑求解三次多项式的问题，然后再将这些结果应用到B样条上去。考虑多项式：

$$g(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{19-142}$$

对它的求解直接包括六个乘法和三个加法。Horner方法重写多项式为：

428 $$g(t) = ((a_3 t + a_2)t + a_1)t + a_0 \tag{19-143}$$

这包括三个乘法和三个加法——一个很大的节省，尤其当我们考虑到多项式可能必须对很多t值进行计算的情况。这个结果可以很容易被一般化到n阶多项式的情况。

如果多项式是在相等间隔区间上求解，$t=0$、h、$2h$、…，那么还有一个方法是可行的，在它的迭代循环中只需要加法。这被称为前向差分方法。假设存在固定增量（h），首先应用到$g(t)$的差分是这样定义的：

$$\Delta g(t) = g(t+h) - g(t) \tag{19-144}$$

同样地，对更高阶可以用下式定义：

$$\begin{aligned} &\Delta^0 g(t) = g(t) \\ &\Delta^i g(t) = \Delta^{i-1} g(t+h) - \Delta^{i-1} g(t) \\ &\qquad i = 1, 2, \cdots \end{aligned} \tag{19-145}$$

通常，对于一个n阶多项式，第一个差分将是一个$n-1$阶的多项式，第二个差分的阶是$n-2$，等等。因此第n个差分将是一个常数(0阶)。对于三次多项式$\Delta^3 g(t)$将是一个常数——即独立于t。根据这些事实我们可以构造一个简单的算法。

应用式（19-145），很容易证明下列结果的正确性：

$$\begin{aligned} &\Delta g(t) = (a_1 h + a_2 h^2 + a_3 h^3) + (2a_2 h + 3a_3 h^2)t + (3a_3 h)t \\ &\Delta^2 g(t) = h^2(2a_2 + 6a_3 h) + (6a_3 h^2)t \\ &\Delta^3 g(t) = 6a_3 h^3 \end{aligned} \tag{19-146}$$

同时根据式（19-145），也容易看到：

$$g(t+h)=g(t)+\Delta g(t)$$
$$\Delta g(t+h)=\Delta g(t)+\Delta^2 g(t) \quad (19\text{-}147)$$
$$\Delta^2 g(t+h)=\Delta^2 g(t)+\Delta^3 g(t)=\Delta^2 g(t)+6a_3h^3$$

因此式（19-146）和式（19-147）暗示下列各项内容：假设$g(t)$、$\Delta g(t)$、$\Delta^2 g(t)$、是已知的（$\Delta^3 g(t)$显然总是已知的，因为它是一个常数）。那么应用式（19-147），$g(t+h)$、$\Delta g(t+h)$和$\Delta^2 g(t+h)$可以渐增地求出。典型地对于曲线渲染的应用，多项式将在范围0到1之间求解，h是按照所需要的增量决定的。式（19-142）和式（19-146）显然可以用来计算开始值 $g(0)$、$\Delta g(0)$、$\Delta^2 g(0)$。剩下的值可以根据式（19-147）渐增地计算出来。这个计算用图19-25中表的形式表述。第一行是预计算。对于每个后续行值的计算是通过对正上方项和右边的一项使用式（19-147），除了最后一列常数以外。第一列给出的是在等间隔参数上所要求的值。注意，整个计算（除了第一行以外）只基于加法运算。

$g(0)$	$\Delta g(0)$	$\Delta^2 g(0)$	$\Delta^3 g(0)=6a_3h^3$
$g(h)$	$\Delta g(h)$	$\Delta^2 g(h)$	$\Delta^3 g(h)=6a_3h^3$
$g(2h)$	$\Delta g(2h)$	$\Delta^2 g(2h)$	$\Delta^3 g(2h)=6a_3h^3$
$g(3h)$	$\Delta g(3h)$	$\Delta^2 g(3h)$	$\Delta^3 g(3h)=6a_3h^3$
…			

图19-25 三阶多项式的前向差分计算

429

自适应前向差分

在前面一小节中我们简单介绍了前向差分的思想。虽然这种方法速度快，但它需要在相等间距的参数值上对曲线进行遍历，如果曲线有一些部分基本上是平坦的，这个方法就不太合适。在此时会需要太多的计算。另一方面，递归细分方法自适应地细分曲线并求出相对平坦的部分，但是这需要付出递归实现的代价。我们希望结合这两种方法，这样就既可以保持前向差分和递归细分方法的简单性优点，同时又可以避免实际做任何递归。

这个问题的解决办法是由Shue-Ling Lien（1987）提出来的，称为自适应前向差分。首先，来看一些背景情况。考虑三次 Bézier 曲线：

$$F(t)=(1-t)^3p_0+3(1-t)^2tp_1+3(1-t)t^2p_2+t^3p_3 \quad (19\text{-}148)$$

这可以用矩阵形式重写成：

$$F(t)=[t^3\ t^2\ t\ 1]\begin{bmatrix}1&0&0&0\\-3&3&0&0\\3&-6&3&0\\-1&3&-3&1\end{bmatrix}\begin{bmatrix}p_0\\p_1\\p_2\\p_3\end{bmatrix} \quad (19\text{-}149)$$
$$=tBp$$

Bézier曲线是用B矩阵表达的。这是定义它为Bézier曲线的工具，因为这个矩阵是表达式中依赖于Bézier公式的惟一部分。

现在假设对参数做一些变换，比如用$a+bt$替换$t(t\leftarrow a+bt)$，那么我们可以再仔细检查这个作为三次多项式求解新曲线的过程，而且重新求解相似的表达式（对于一些矩阵C）：

$$F(a+bt)=tCp \quad (19\text{-}150)$$

因为Bernstein函数形成一个基，我们知道任何多项式可以重新表示成 Bézier 形式，现在可以再次看到这一点。假设能求出矩阵A，这样：

$$BA=C \tag{19-151}$$

将此替换入式（19-150），得到

$$F(a+bt)=tB(Ap) \tag{19-152}$$

所以点Ap是 Bézier 点，对应于由式（19-150）所给定的曲线。

考虑由下式给出的参数变换：

$$L:t\leftarrow\frac{t}{2} \tag{19-153}$$

430 它描述的是曲线在参数范围 $[0,\ \frac{1}{2}]$ 上的部分。

我们已经知道如何通过使用 de Casteljau 算法构造对应于曲线部分的Bézier 控制点，所以在这个特殊情况中的矩阵A将确定点Ap集合是曲线在$[0,\ \frac{1}{2}]$上的Bézier点。对于（R）有一个相似的结论：

$$R:t\leftarrow\frac{t+1}{2} \tag{19-154}$$

它表示曲线在参数区间$[\frac{1}{2},\ 1]$上的部分。

通常对任何曲线，这两个变换L和R给出了曲线在 $t=\frac{1}{2}$ 点处按照 de Casteljau细分所得到的左段和右段。Shue-Ling Lien等(1987)意识到进一步的变换：

$$E:t\leftarrow t+1 \tag{19-155}$$

即

$$R=EL \tag{19-156}$$

首先应用 L、然后应用E，得到结果R。

由应用L和R所生成的点集合对应于de Casteljau 递归细分中获得的点。然而，使用（19-156）是可以避免递归的。自适应前向差分算法是这样的：

（1）应用L足够多次数以便得到曲线的一个小段（例如曲线在1个像素里，或者控制点大约在一条直线上，又或者没有什么准则）。

431 （2）应用E得到曲线上的后续点，当准则为真的时候（这是算法中前向差分的部分）。如果曲线段变得太小，那么应用L^{-1}。当准则不再为真，那么应用L。

这个方法需要对操作E和L有一个离散的表示。从上面知道，因为它们是参数变换，我们可以求出一个矩阵A，由它来转换控制点。事实上$A=B^{-1}C$。所以三种运算事实上对应于三个A矩阵，使用它们转换控制点。

显然可以直接使用这个方法于Bézier曲线。然而，E、L和L^{-1}所对应的矩阵不是特别适合于计算。取代Bernstein基函数（对于Bézier曲线），Shue-Ling Lien 等（1987）提出了所谓的前向差分基，由此产生了高效计算的矩阵。正如下列所定义的：

$$
\begin{aligned}
D_0(t) &= 1 \\
D_1(t) &= t \\
D_2(t) &= \frac{1}{2}t(t-1) \\
D_3(t) &= \frac{1}{6}t(t-1)(t-2)
\end{aligned}
\qquad (19\text{-}157)
$$

因为这是一个基，最多为三阶的任意多项式函数可以写成下列形式：

$$
F(t) = p_0 + p_1 D_1(t) + p_2 D_2(t) + p_3 D_3(t) \qquad (19\text{-}158)
$$

把它作为一个多项式在幂基中展开，有：

$$
F(t) = t^3\left(\frac{p_3}{6}\right) + t^2\left(\frac{p_2 - p_3}{2}\right) + t\left(\frac{p_3}{3} - \frac{p_2}{2} + p_1\right) + p_0 \qquad (19\text{-}159)
$$

现在做替换 $L: t \leftarrow \dfrac{t}{2}$ 得到：

$$
F(t) = \frac{t^3}{8}\left(\frac{p_3}{6}\right) + \frac{t^2}{4}\left(\frac{p_2 - p_3}{2}\right) + \frac{t}{2}\left(\frac{p_3}{2} - \frac{p_2}{2} + p_1\right) + p_0 \qquad (19\text{-}160)
$$

我们想要将式（19-160）重写成式（19-159）的形式，以便找回相同的基，即求出点集 q_0、q_1、q_2、q_3，这样有：

$$
F(t) = t^3\left(\frac{q_3}{6}\right) + t^2\left(\frac{q_2 - q_3}{2}\right) + t\left(\frac{q_3}{3} - \frac{q_2}{2} + q_1\right) + q_0 \qquad (19\text{-}161)
$$

与式（19-160）相等。我们可以对 t 的各幂项的系数划等号，这样可得到：

$$
\begin{aligned}
q_0 &= p_0 \\
q_1 &= \frac{p_1}{2} - \frac{p_2}{8} + \frac{p_3}{16} \\
q_2 &= \frac{p_2}{4} - \frac{p_3}{8} \\
q_3 &= \frac{p_3}{8}
\end{aligned}
\qquad (19\text{-}162)
$$

最后可以写成矩阵形式：

$$
L = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \frac{1}{2} & -\frac{1}{8} & \frac{1}{16} \\
0 & 0 & \frac{1}{4} & -\frac{1}{8} \\
0 & 0 & 0 & \frac{1}{8}
\end{bmatrix}
\qquad (19\text{-}163)
$$

432

可以由此计算出下式：

$$
L^{-1} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 1 & 0 \\
0 & 0 & 4 & 4 \\
0 & 0 & 0 & 8
\end{bmatrix}
\qquad (19\text{-}164)
$$

相似地，分析可得：

$$E=\begin{bmatrix}1&1&0&0\\0&1&1&0\\0&0&1&1\\0&0&0&1\end{bmatrix} \tag{19-165}$$

检查这些矩阵中的值，它们的使用只包括加法和指数为2的乘法。因此它们在实际应用中是非常有效率的（同样对于硬件也是这样）。

因此，使用自适应前向差分来渲染曲线，首先必须将它用前向差分基来表示，然后应用矩阵E、L和L^{-1}来产生曲线上的点。

19.15 小结

在这一章中我们介绍了计算机辅助几何设计中的一些主要的成果。我们通过开花的方法，由此导出Bézier曲线，并考虑了这些曲线的性质以及de Casteljau 细分算法。将这一般化到B样条曲线，这里曲线段的序列可以被端点到端点地连接在一起，以形成一个比较复杂的曲线，
433 在接合处给定连续性约束。我们说明了开花方法如何可以用来导出B样条基函数，这是对B样条曲线定义的更一般方法。一旦理解了曲线的思想，也就可以将它非常容易地过渡到曲面上来。我们介绍了Bézier和B样条张量积曲面，以及三角形Bézier面片。使用开花方法于B样条来产生曲线插值的特殊结果：假设曲线上的一组数据点，求出产生这样一条曲线的控制点。最后介绍了渲染多项式曲线的一些有用的基本思想——前向差分和自适应前向差分。

第五部分 虚拟环境中的动态特性和交互

第20章 虚拟世界中人的动态特性

20.1 引言

至此我们已经了解了虚拟环境的渲染。然而，对于绝大多数实时系统来讲，一个主要的部分是人和虚拟环境之间的交互问题。我们通常使用“用户”这个术语描述在虚拟环境中交互的人。由于在沉浸式虚拟环境的上下文中这些人是虚拟环境的一部分，参与到了环境中（连同其他人一起），因此这个词并不适当，我们倾向于用“参与者”这个术语，而不是“用户”。在虚拟环境中最基本的交互能力是在参与者的控制之下移动虚拟照相机镜头。这个能力也称为*探索导航*或*运动*。提供这种运动能力的实时系统通常叫做*漫游系统*，因为参与者不能触摸对象或影响系统进展，只能观察它。

更具交互性的系统允许参与者与单个对象进行交互，激活它们的内在行为或改变其属性。
在这本书中，我们不讨论诸如3D 菜单和小部件（widget）之类抽象的用户界面设备，而是专 434
注讨论如何去触摸、选择或操纵对象。

参与者如何定义和激活交互是一个非常复杂的过程，在很大程度上依赖于所使用的输入和输出设备。我们从介绍*虚拟现实模型*开始，也就是说，我们是在参与者完全沉浸在计算机显示并能在虚拟世界中直接操纵对象这样的人机界面风格中进行讨论。这些技术对场景建模、交互和行为有很高的要求。为此，我们将集中研究如何在虚拟环境里表示参与者，以及为使这种表示变得逼真所需要的跟踪和感知技术。

其次我们将讨论在当前系统中所受到的一些限制，包括硬件和软件两个方面的问题，还要花一些时间概略介绍一些有关*碰撞检测*的问题。下一章将描述更一般的交互技术，即在当前*非沉浸式系统*中通常所使用的一些技术。

20.2 虚拟现实模型

虚拟现实系统有一个特定的目标，那就是让参与者相信他们确实位于感官系统所显示的环境中。这种对于环境的“存在”感称作*存在*或*遥现*，如第1章所讨论的（Held and Durlach，1992；Draper et al.，1998）。产生存在感的环境通常等同于一个凭直觉来使用的环境和一组自然的交互隐喻。如果参与者存在于一个逼真方式建模的环境中，那么我们能预期到参与者有在相似真实情形中的先验知识。相反，如果这样的环境不能实现参与者对行为和交互的期待，则存在感将受到破坏。举例来说，在一个社会集会的仿真环境中，如果对方伸手希望跟你握手，而自己却不能伸出手回应的话，这样的环境就很难有存在感。此外，参与者对其未能实现的预期行动所做出的反应将会在未来的遭遇中导致恐惧或混乱。

在虚拟现实模型中，显示试图通过视觉、听觉、触觉、嗅觉和味觉信息将参与者完全包围住。Ivan Sutherland的"ultimate display"（Sutherland，1965），以及更知名的《星际迷航》中的Holodeck对现实的仿真是如此逼真，以致于物理伤害可能降临到参与者的头上，但这些都是未能实现的雄心。

应该指出的是，绝大多数虚拟环境显示系统专注于视觉系统。的确，这本书掩饰了作者的个人倾向，我们在这一小节之外不考虑声音。然而，根据经验，在某些情形下，声音的存在是必要的，存在适当的背景声音和环境提示能增强存在感。听觉有视觉所不具备的一些独特性质，声音提示可以定位于任何方向，而且听觉系统有能力分辨出多个并存音频流中的某
435 个声源（Wenzel，1992）。

最后要提到的是，极少数系统使用了某种形式的触觉、嗅觉和味觉显示。对于这种系统的研究还在继续，但是存在重大技术障碍的这些显示有基础性问题没有解决（例如显示会涉及到大面积皮肤）。我们推荐对此感兴趣的读者阅读Kalawsky的综述文章，他对这些显示的生理需求以及当前所采用的一些技术做了比较全面的介绍（Kalawsky，1993）。

沉浸感

在虚拟现实模型中，首要的需求是让参与者沉浸在显示系统中。我们将沉浸感看成是一些显示属性的组合。首先，显示的信息包围了参与者。举例来说，显示尽可能地覆盖视觉范围，当参与者环顾左右的时候，他们不会什么也看不见。显示这个概念是广泛的，它包括了来自多种感知通道的信息。显示是相容的，那些分散注意力的信息（例如真实性） 被排除在外。最后，显示是生动的，也就是说它有很高的分辨率，是有丰富色彩的，同时覆盖了听觉的全部范围，正如第1章所介绍的。

典型的沉浸式系统包括头盔显示器（Melzer and Moffitt，
436 1996）和Cave自动虚拟环境 （CAVE[⊖]）（Cruz-Neira et al.，1992，Cruz-Neira et al.，1993）。在头盔显示器中通常有一对显示器，分别对应两只眼睛，相对于参与者的头部固定。除此之外的视野被某种形式的面罩所屏蔽。图20-1所示的是2000年典型的中型HMD，称为Virtual Research V8。每块屏幕具有800 × 600个颜色像素。头部被跟踪以便能够动态地调整虚拟照相机的属性，这些属性定义了左右眼的图像。我们在之前曾介绍了立体渲染的概念。读者可以进一步参考 McAllister（1993）以及Melzer and Moffitt（1996），了解关于头盔显示器的设计。

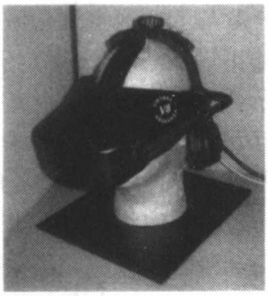


图20-1 Virtual Research V8 头盔显示器（由Virtual Research Systems公司提供）

CAVE这类显示器是沉浸感的另一种极端：一组大显示表面包围了参与者，同时参与者可以在一个确定的空间范围内自由走动。彩图20-2中给出的是位于瑞典斯德哥尔摩皇家工学院（KTH）并行计算机中心（PDC）的VR-CUBE，这是第一批六面的CAVE显示器之一。

虽然CAVE和HMD系统的目标是让参与者沉浸在视觉系统中，但是它们实际上具有相当不同的属性。在CAVE显示器中参与者能看见他们自己的身体，而且对象不能出现在身体的前

⊖ CAVE是伊利诺伊州立大学董事会的商标。此术语还通常作为这种显示器的总称。

面。举例来说，因为手会遮挡显示屏幕，所以任何试图跟踪手并用手摆放虚拟对象的努力都将失败。在HMD中看不到真实的身体，而是创建了一个虚拟的身体。因为整个显示是计算机产生的，所以排除了近处空间中对象受遮挡的情况。要知道我们并没有介绍所有的虚拟现实系统，比如说增强现实系统，这是在HMD中将现实和虚拟构想混合在一起的一种系统（Azuma，1997）。

在一个沉浸系统中，显示就是要尽量提供给感知系统一个比较宽的输入范围。然而，沉浸感只是指显示属性而不是所显示的内容。沉浸式系统完全有可能显示超现实的东西或者是完全幻想的世界。对于设计者和程序员（包括图形程序员）来讲，还是要使得所设计的世界看起来与现实保持一致和可信。

人机界面

人和桌面计算机之间的交互包括参与者使用鼠标和键盘来执行动作，激活各种窗口、图标和菜单。这种界面存在两个问题：形成适当的行动来执行一项任务的困难，其次是对响应的理解和评估的困难。前者是执行的障碍，后者是评估的障碍（Hutchins，1986）。这也就是说，在界面上找到某些有效的功能通常是困难的，即使当它已经被激活，可能也没有立即的反馈，即使有，它也可能是不容易理解的。为了减轻这些问题，通常界面设计者使用直接操作范型，世界模型构造在其中，对象可以被参与者直接移动、选择或编辑（Hutchins et al.，1986）。这样的例子包括所见即所得（WYSIWG）风格的文档编辑器和计算机辅助设计程序包。

直接操作风格遍及整个虚拟现实模型。的确，在一个层次上参与者自身就是界面，因为三维显示完全将参与者包围起来，而且他或她得到了很好的跟踪。举例来说，当使用立体头 437
盔显示器装置时，显示器相对于参与者的头部是静止的。对头部跟踪产生视图的虚拟照相机的运动直接对应于参与者的头部运动。这种对平移和旋转的一对一的映射是一个非常直观的映射，虽然根据我们的体验，人们在第一次戴上HMD时并没有意识到头部被追踪，一旦了解了这一点（或被告知可以“四处看看”），他们都不会忘记该如何转动视图。

把它与桌面情况进行比较。摆放照相机是一个自由度为6的任务。为了利用鼠标和键盘来完成这样一项任务，我们必须构造一些映射获得多个独立控制维度并将它们结合在一起。实现这一点的典型方式有两种：要么将不同的设备映射到不同维度（例如鼠标旋转照相机，键盘上的方向键映射为相对于当前照相机的位置平移），要么构造界面通道（例如使得鼠标运动映射为在 XZ 平面上的平移，如果左边的 Shift 键被按下，那么鼠标运动映射为在 XY 平面上的平移）。下一章我们将详细讨论一些常见的交互技术及其实现问题。

在虚拟现实模型中的照相机控制任务应该比用上面描述的技术具有更低的认知负荷。而且在虚拟现实模型中我们两只手是自由的，可以腾出来做其他的任务，但在桌面条件下，我们需要使用一只手或两只手来操作键盘和鼠标。这是虚拟现实模型能力之一，它尽可能密切跟踪参与者的身体，从而将输入控制的机会最大化。对于虚拟现实系统，通常除了参与者的头部被跟踪以外，参与者的手也同样得到跟踪，所以拾取和交互任务都变得同样容易。

虚拟现实模型中的交互

一旦参与者是沉浸式的，并得到精确跟踪，我们就假定输入和输出是已注册的。注册暗示参与者四肢相对于彼此的运动得到正确的测量。如果做到了这一点，那么参与者能预期系

统的结果。举例来说，如果他们将自己真实的手放在真实的脸前面，虚拟手或光标将会出现在视图中正确的位置。

我们已经看到在桌面系统中一些可用性问题迎刃而解了。举例来说，当操作一个窗口系统时由于屏幕的混杂可能失去鼠标光标，但是我们很难想像在虚拟现实系统中参与者会失去对他们手的控制。这一点说明执行障碍在虚拟现实模型中是比较低的。然而，必须注意到这种障碍依然是存在的，因为虚拟环境如何建模以及它将如何反应都存在很多不一致性。

438 虚拟现实系统使评估障碍变小了吗？让我们看一下挥动手的例子。如果参与者看见虚拟的手在挥动，他们就很容易评估发生了什么，并建立结果和影响之间的联系。然而，如果看见的是光标，也许可能感到困惑，因为他们预期看见的是一只真实的手。确实有证据可以表明虚拟现实系统不仅是需要一只手，一个完整的虚拟身体更有用（Slater and Usoh，1994；Mine et al.，1997）。我们最初问题的答案因而还是不确定的。在一般情形中，通过在虚拟环境系统中给出离奇和美妙的环境构造，我们可以认为沉浸感给予参与者对三维空间的一个很好的理解。作为虚拟环境的设计者，我们能在虚拟环境的设计中使用三维手段来达到应有的效果。换句话说，如果所构造的桌面虚拟环境不是很好，那这样的系统将是难以理解的，同时也是不容易操作的。参与者应该可以建立一个认知模型，来描述世界的运行规律，以及如何与这样的世界进行交互。这并不意味着应用都得非常逼真地建模，但是这经常发生，因为对于天真的参与者来说，这样做很容易让他们理解他们正在经历的事情。

20.3 人体仿真

从先前一小节的讨论中我们可以明显地看到，对于任何虚拟环境系统来讲，一个重要的组成部分是参与者在该环境中的模型。在某种意义上讲，人体模型是虚拟环境系统的界面描述——眼睛是视觉界面，耳朵是听觉界面，手是执行单元。如我们已经看到的，给出根据自身观点所提出的人体几何描述是非常有用的。这不仅仅是为了展现参与者的身体，对手和手指的几何描述还将构成拾取对象真实仿真的基础（Boulic et al.，1996）。

在对虚拟现实模型的描述中，我们都将参与者看成是完全沉浸其中的，而且得到很好的跟踪，所以他们身体的运动映射为显示中的一致性变化。对于运动，如果忽视跟踪系统的某些限制，例如有限的范围，那么虚拟现实模型允许我们避免“奇妙的”交互隐喻，因为参与者能自由地交互。

人体模型的构造

现在已经存在能跟踪几十个点的非常复杂的跟踪系统。例如，Ascension MotionStar能跟
439 踪多达90个点。这种系统最普遍用在脱线动画中的运动捕捉，而很少在沉浸式显示中使用。但是如果用在交互式环境中，拟人模型中的这些点可以单独得到跟踪。为了实现跟踪系统，存在一些非常复杂的骨架层次结构，而且对于人的骨架已经有了一个初现的标准H-Anim（H-Anim，1999），如图20-3。

一种比较典型的人体建模情形是对头部、躯干和两只手跟踪。我们能创建一个分离部件组成的非常简单的化身模型，但是最好能从这些有限的传感信息中确定身体一些关节。

从有限的跟踪器数据推导出多关联肢体位置是*逆向运动学*（inverse kinematics）问题的一个例子。因为骨架有很多自由度，因而就有多种方式来配置它，这样所跟踪的肢体就能位于

它们应该在的位置上。举例来说，假设已知肩膀和手的位置，肘部的位置就只有有限个数目。发现一致的约束是一个问题。继续这个例子，也许让肘部总悬于最低的位置处看起来像是个明智的做法，但是这会在视觉上产生不正确的结果，尤其对于某些运动，例如将手从脸前方移动到头部后面。Badler et al.（1993）对这个问题给出了四个跟踪器情形的解法。

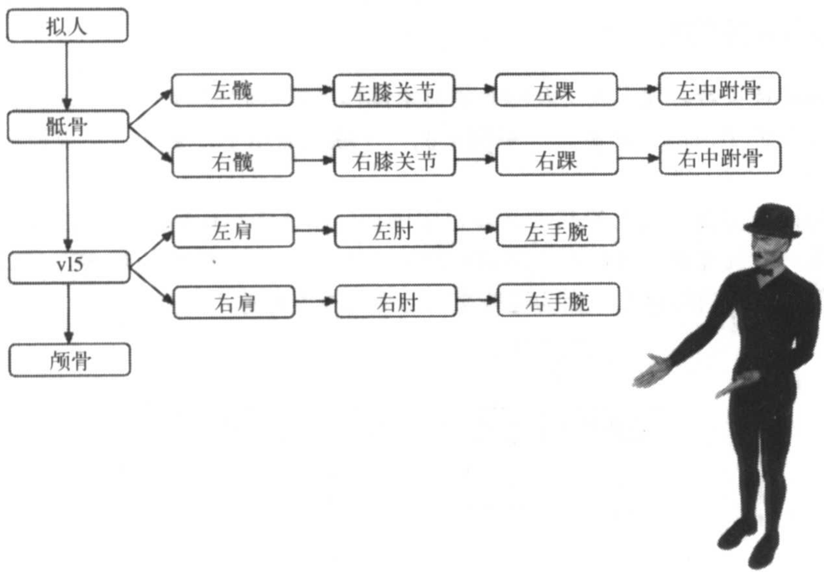


图20-3 摘自H-Anim的人体层次结构，以及一个运动捕捉数据所驱动的化身（由Daniel Thalmann主管的位于瑞士洛桑的计算机图形实验室（LIG）提供，人体动画制作由Christian Babski完成，人体设计由Mirelle Clavien 完成）

对参与者的跟踪

有许多不同的跟踪技术正在使用中（Meyer，1992）。Polhemus Fastrak（Polhemus，HTTP）是一个普遍使用的电磁跟踪设备，它由一个产生变换磁场的基发射器组成。这个磁场引发接收器设备中产生电流，从这便可以导出接收器相对于发射器的位置。电磁跟踪器的一般问题是它们受到来自环境中金属的干扰以及来自其他电子系统的干扰。 440

由于有这些干扰问题，跟踪系统通常有一个有限的操作范围。商用系统（例如Ascension MotionStar）能支持一个大约边长为10米（Ascension，HTTP）的立方范围，但是更常用系统一般使用范围只有几米边长的立方范围。

视觉、超声波和惯性的跟踪技术也是存在的，但是这些技术的应用不是很普遍，主要原因是由于这些技术不太令人满意，或者价格过于昂贵。比较有前景的研究是采用混合技术，例如Intersense的跟踪器混合使用了惯性和超声波技术（Intersense，HTTP）。

对不同跟踪系统的选择主要根据五个参数：精度 （包括角度和位置）、分辨率、范围、总的系统延迟和更新频率。精度、分辨率和有效范围的含义是明显的。系统延迟和更新频率的含义更复杂一些。总的系统延迟是指从参与者做出行动（比如一个头部运动）到屏幕上出现图像之间所用的时间。这个延迟在沉浸式系统中是非常重要的，因为它影响参与者实现电机控制的可靠度。比如，如果虚拟手严重滞后于真实手的动作，参与者会难以自然地触摸对

象，因为他们掉入等待视觉反馈的情形中，即在进行下一个动作之前让视觉反馈跟上所做的动作。如果更新频率不够快的话，场景就不能提供给参与者应有的连续性。形容这个参数的一个典型数据是所谓的帧频，也就是说跟踪器的更新频率不应该低于15Hz（Barfield and Hendrix，1995）。

20.4 与虚拟人体的交互

跟踪的限制意味着必须引入交互隐喻到对象操作（抓取和移动）以及移动（运动）。还有其他限制需要我们引入隐喻的概念。举例来说，在触觉显示中，参与者在虚拟环境中不能走入对象内部。

交互隐喻很难实现，因为参与者必须记住该如何激活隐喻中所暗含的技术。即使可以使
441 显示具有非常高的沉浸感，我们距离所谓理想的"终极显示"，即虚拟现实与真实现实难以区分的环境，仍然有相当的距离，因为这些交互隐喻很少匹配适当的真实世界任务。

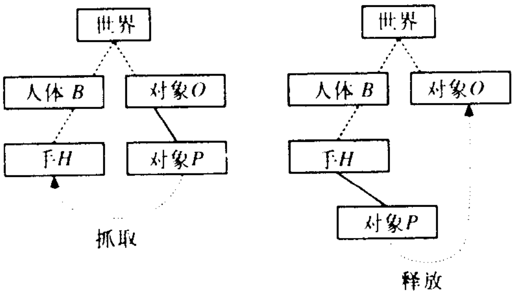


图20-4　在虚拟现实模型中指定对象操作

我们将会在下一章中较深入地讨论交互隐喻问题，我们要放宽对参与者人体跟踪的需求，并且在讨论中将依赖更抽象的输入设备。

对象操作

通常在对象操作体验中遇到的首要问题是手一般只有一个点得到跟踪，因为手势没有获得，所以虚拟手只有静态的形状。这使得抓取手势很难执行，通常的交互隐喻是通过虚拟手触碰虚拟对象来产生抓取意图，然后按一个手持式设备上的某个按钮完成抓取动作。这种抓取隐喻的难易程度依赖于按钮设备的形式。这些设备差异很大，较为简单的如在球上安装的按钮（Polhemus公司的 3BALL）；较为复杂的是数据手套，如Pinch Glove（Fakespace，HTTP），当参与者的拇指和其他手指接触的时候，就表示按下了一个按钮。

抓取交互隐喻的一个必要部分是测试虚拟手和其他虚拟对象之间的相交。一旦确定了一个要抓取的对象，就可以将它贴于手上，当移动手的时候它就将跟着手移动。内部的实现过程是将要移动的对象从它在场景图中的当前位置取下来，当对象被抓取的时候，我们就把它作为手的一个子节点。在抓取的过程中，对象的位置受到手的全局坐标变换的影响，这样才能跟随它一起移动。当对象被释放的时候，它就会从手层次结构中离开，并再次回到原先场景图中的位置。

这个过程用图20-4可以说明。第一步包括计算从手到对象的相对变换M_R。这是由下列等式给定的：

$$M_R = (M_B \cdot M_H)^{-1} \cdot M_O \cdot M_P \tag{20-1}$$

这里M_B是从人体到世界坐标的变换，M_H是从手到人体坐标的变换，M_O是从对象O的坐标到世界坐标的变换，M_P是从对象P到对象O坐标的变换。 442

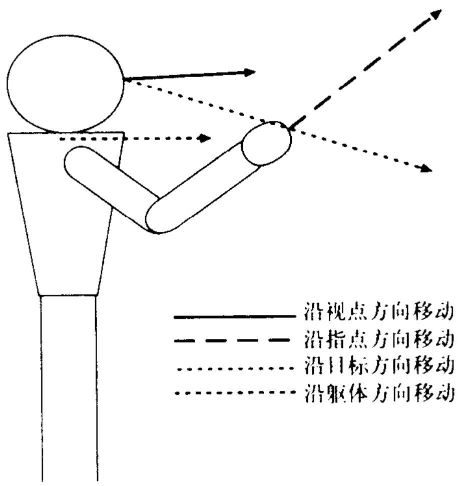


图20-5 指定移动的方向

在操作之后对象$M_{P'}$新的局部变换可以用下式计算:

$$M_{P'} = M_O^{-1} \cdot M_B \cdot M_H \cdot M_R \tag{20-2}$$

Robinett 和 Holloway（1992）给出了执行这种风格的操作的第二种方法，该方法不需要在场景图中移动和重新确定对象的位置。

移动

跟踪器范围限制意味着距离超过了一米或两米远的时候，参与者必须使用一个交互隐喻来移动。常见的做法是用手持式设备上的第二个按钮作为移动的开始切换键。移动的方向可以取为视点的方向或者是手所指向的方向，抑或是躯体所朝向的方向，见图20-5。速度可以是固定不变的，也可以由参与者来控制。对于沉浸式情况的其他例子请参见Mine et al.（1997）。

我们也经常将参与者的移动限制在水平平面上，因此他们的眼睛高度保持为常数。当在凹凸不平的表面上移动时，就会出现问题。我们还不能够（到目前为止）显示不同高度的地面，所以障碍物要么阻止了移动，要么参与者基于一些启发式算法上下地跳跃到不同的新表面上继续移动。在这里，我们需要某种碰撞检测形式，而且也需要有关参与者的人体模型。举例来说，如果对象碰到的是参与者膝部以下的部位，那么参与者就向上移动，这样他或她的脚落在障碍物的上面，否则，参与者向后移动身体让障碍物仅仅碰到身体。 443

对移动问题的相关研究有了很多进展。线性的（Brooks，1986）和全向的踏旋器（omni-directional treadmills，参见Darken et al.，1997）都被当作输入设备使用。这些都直接解决跟踪器范围问题，但是它们都有严重缺点，不适合广泛应用。

以身体为中心的导航

一旦参与者能够在环境中运动，一个显而易见的问题就提出来了，即他们的虚拟身体如

何与环境交互。除在先前的一小节中提到的表面跟踪之外，阻止参与者的虚拟身体部分或全部穿墙而过通常也是很需要的，因为这些容易让人产生不真实感。这需要检测参与者的身体和虚拟环境之间的碰撞，所以当身体碰触到一个障碍物的时候，虚拟世界中的移动就会被停止。然而这个问题立即变得复杂起来。如果一个参与者朝一面墙壁走去，然后伸出自己的一只手臂，我们可能不得不将他的手臂躲开墙壁以避免手穿入墙壁。许多系统为避免发生这个问题，采用了一个大的包围盒包围住参与者，使得参与者与墙的距离始终大于手臂的长度，但是这也有问题。因为这些技术中任何一项都会造成参与者的困惑，因为从物理上讲，阻止参与者执行那些将导致碰撞响应的物理行动是不可能的。在下一小节中我们将会概略说明可以作为检测身体和场景之间碰撞测试的一些技术。

20.5 对象间的碰撞检测

虚拟人与虚拟环境交互的一个重要部分是对象间碰撞测试功能。我们解决这个问题的基本步骤有两步，首先给出当两对象相交时的穷举测试方法，然后说明在现实中如何尽可能地避免做这种测试。

穷举测试

我们假设进行碰撞检测的所有对象都是由三角形构成的。设一对对象，一个是由m个三角形组成，另一个由n个三角形组成。如果有一个可靠的三角形-三角形相交测试，我们就能通
444 过对三角形对进行穷举测试得到两对象间的相交结果。这需要 $m \cdot n$次三角形-三角形测试，我们先叙述三角形-三角形相交测试算法，然后简要地列出各种优化方法。

下面的方法由Moller（1997）给出。在两个三角形A和B的比较中：

（1）如果三角形A的所有顶点完全位于包含 B 的平面的一侧，则排除三角形A和B（它们不相交）。

（2）否则，包含A和 B的平面必然相交于一条直线 L。

（3）求出L与A相交的线段部分（L_A）和L与B相交的部分（L_B）。

（4）A和B相交当且仅当L_A与L_B有重叠。

这个过程由图20-6说明。

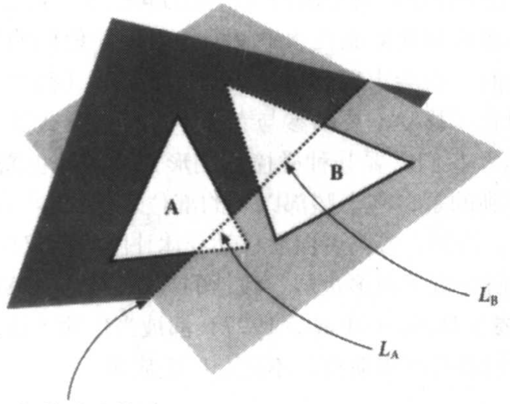


图20-6 三角形-三角形的相交测试例子

这个测试对于大量对象的情况显然代价很大，虽然有方法可以加速这个测试，但是需要投入很多注意力来寻找快速排除测试，通过这种排除测试迅速确定两个对象是否不可能重叠。

基本排除测试

最简单的对象排除测试是基于距离的测试。每个场景元素由一个包围球包围。如果两个包围球中心之间的距离大于两包围球半径之和，则两对象不可能重叠。 445

这个测试执行起来十分简单，但是在使用中又是非常保守的。更精细的测试是分割平面测试。如果能找到一个平面，使得其中一个对象上的所有点全部位于该平面的一侧，而另一个对象上的所有点全部位于平面的另一侧，则这两个对象根本不会发生碰撞。这个方法的关键是求出一个理想的分割平面。

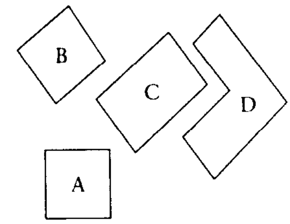


图20-7　碰撞对象的排除测试例子

图20-7给出了一些例子。对象A和B不重叠，因为有一个与轴对齐的平面将它们分离开来。对象B和C不重叠，因为由C的一条边形成的一个平面将B和C分离开来。对象C和D也不重叠，但是没有分割平面。此时我们一定要回到穷举测试。

包围盒范围测试

如果对场景元素构造了轴向对齐的包围盒，那么我们就可以得到另外一个简单的排除测试。包围盒由x、y和z三个方向范围确定。一个重要结论是两个包围盒在三维中重叠当且仅当它们分别在x方向、y方向以及z方向上重叠。相反，如果在x、y和z方向上任何投影不发生重叠，两包围盒不重叠。图20-8说明了这一点。我们能看到元素A和B在x轴方向上重叠（$A_{xmax}>B_{xmin}$），但是在y轴方向上不重叠（$A_{ymax}<B_{ymin}$）。元素A和C在任何轴向上都不重叠，所以可以不用对它们进行X或Y轴测试。元素B 和C在两个轴向上都重叠，所以它们是碰撞的候选对象。

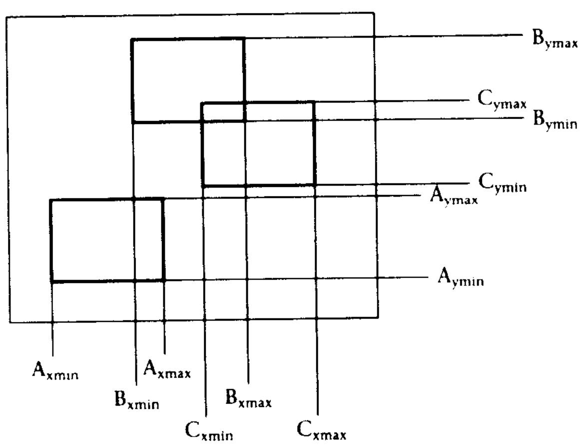


图20-8　用一系列一维重叠测试来进行碰撞检测

20.6　一般性碰撞检测

检测一组对象之间的相互碰撞问题比检测一对对象的情况要麻烦得多。对于n个对象，就

有n^2对对象有可能需要进行重叠测试。通常，我们所采取的策略是求出一个空间分割，以便
446 尽可能多地丢弃无需检测的对象对，为此需要使用第16章所讨论的在加速光线跟踪上下文中的一些技术。对于所有剩下来的对象对，还要使用前一小节中的某些技术来确定是否对象之间真的发生了碰撞。

我们曾介绍了为加速光线跟踪的"一致空间细分"概念。现在可以使用相似的技术来加速碰撞检测。

对于场景我们构造一个轴向对齐的包围盒将其包围住，并沿每个轴向上进行一致的细分。在所产生的每个细分单元中，用一个链表记录下与此单元相交的所有场景元素。这通常是用场景元素的包围盒来代替的。这样，对于每个其链表包含超过一个元素的单元，我们必须对链表中的对象进行彼此测试来检查是否真的发生了重叠。一旦一对对象被测试过一遍，就没有必要再次测试它们，所以要保留一个链表来保存所有经过测试的对象。

这个过程可以用图20-9来说明。元素A和 B 共享一个公共单元，并且对象对是重叠的。然而，元素C、D和E 也共享一个公共单元，但是CD、DE、EC彼此之间都不发生重叠。

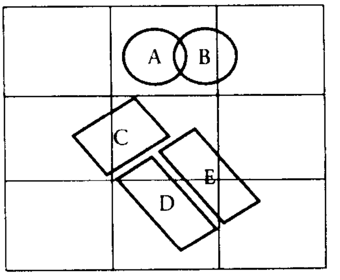


图20-9　用常规空间细分方法求出可能的碰撞对

显然，选择更密的空间细分发现不可能发生重叠的对象对的机会就会增加。其代价是要增加内存的使用和更加昂贵的单元链表的维护，尤其当对象是动态的时候。当然我们也可以使用与八叉树、BSP 树或层次包围盒结构应用相似的方法。

20.7　有关VRML的注解

本章前面曾经提到过VRML97里面描述人体模型的 H-Anim 标准。需要注意的是，H-
447 Anim化身的使用限制为对参与者的视觉描述，化身不能用来作为本章所介绍的交互的基础。VRML97本身确实提供了设施来描述能在移动中处理碰撞检测的非常基础的虚拟人体。我们会在下一章中讨论它。

20.8　小结

我们通过交互的虚拟现实模型介绍了虚拟环境的人机交互。如果对参与者建模并跟踪他们的运动，那么就能够使用如手势和触觉等技术作为输入手段。相反，我们能够定制渲染和其他显示特性来营造让参与者沉浸其中的氛围。

因为显示的形式、跟踪的范围或精度、显示系统中的延迟以及程序设计模式的限制，采用当前的技术这项研究是很有局限的。

我们也介绍了对象之间的碰撞检测问题。首先描述了两个对象之间的碰撞检测，其次讨
448 论了如何通过空间细分方法尽可能避免对象间不必要的测试。

在下一章中我们将研究交互的另一方法，即在非沉浸式系统中经常使用的交互方法。

第21章 实时交互

21.1 引言

虚拟现实模型中的交互技术是实时系统交互方法的一个极端。在这一章中我们将换一个角度，不从用户模型出发，而是从简单交互任务的角度以及用户如何使用通常的交互设备执行这些任务的角度来分析问题。

对于一类实时3D交互模型，用户坐在显示器前面，使用键盘、鼠标或其他限制在桌面上 449
使用的交互设备，有时我们称之为*桌面虚拟现实*，也称*鱼缸虚拟现实*或*非沉浸式虚拟环境*。这些术语涵盖了很大一部分系统，包括众多的设备和显示类型。

在这一章中我们将讨论一些常用的交互设备，接着说明如何使用这些设备来执行基本的交互任务，包括选择、操作和移动。

21.2 桌面交互设备

一个交互设备能识别用户的某些物理行为，例如转动一个刻度盘，并在某个范围内以某个精度报告这一行为。举例来说，一个刻度盘可以报告在 0 到 99 之间的一个整数值，代表从 0 到2π弧度，即一个完整的旋转范围，那么虚拟环境软件就必须能够映射这个值为一个镜头旋转，比如映射为镜头绕局部坐标系的Y轴旋转。交互设备的设计所涉及的内容是十分广泛的，对此的全面介绍超出了本章的范围。读者可以参阅 Foley et al.（1984）、Buxton（1986）and Mackinlay et al.（1990）。

沿用Mackinlay等给出的分类法，并使用来自Buxton 等给出的例子，一些代表性的设备如图21-1中所示。

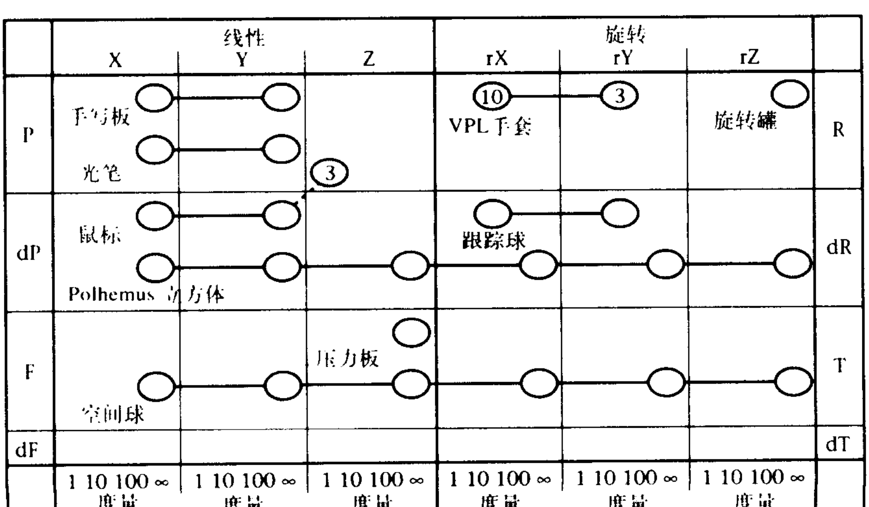


图21-1 Mackinlay等对一些输入设备的分类

这个分类法将设备拆分成它们所感知的单个部分，对这些部分进行分类并重新组合，由此构成关于设备的完整描述。图表中的每个圆圈代表一个物理属性的传感器。分类使用下面列出的物理行为的种类，对于每个行为给出从离散响应到连续响应粒度的一个大致概念。

分类包括：

- 线性/旋转
450 • 定位、旋转/力、转矩（P、R/F、T）
- 相对/绝对
- 方向
- 灵敏度（1=离散的、10=小范围、100=大范围，∞=连续范围）

这样，一个简单按钮就是一个传感器，可以分类为在Z方向上线性的、固定的、绝对的且灵敏度为1。也就是说，它感知一个离散二进制值[⊖]。

在实际中一个单一设备可能感知多个量，这样的设备可以看成是由比较简单的一维传感器组成（Mackinlay et al.，1990）。举例来说，鼠标是由下面这样几个传感器组成：在X方向报告连续量的线性定位传感器、在Y方向报告连续量的线性定位传感器、以及通常都有的两个或三个上面所描述的简单按钮。这里给出三种组成类型。

第一种类型是*合并组成*。两设备的合并组成产生一个单一设备，能同时生成两个初始设备的合并域中的输出。组成的第二种类型是*布局组成*。多个设备的布局组成产生一个能独立感知每个设备独立属性的单一设备。按钮面板是这样组成的，因为每个按钮可以独立感知并可以独立操作。第三种类型是*连接组成*。一个很好的例子包括位于小部件里的滑杆这样的虚拟设备，它将鼠标的输出映射到第二个设备的输入，而第二个设备输出一个不同域中的值。

在图21-1中，合并组成是用实线指示的，布局组成是用点划线指示的。因此手写板或光笔就是合并两个绝对定位感知的设备，一个为X方向，另一个为Y方向。鼠标是合并了两个定位传感器的合并组成和带有2~3个按钮的布局组成。

在图21-1中还有一些其他的流行设备，对于绝大多数设备都有若干个变种，特别是在有可能增加按钮的地方。

Polhemus Fastrak和Spaceball 2003是两种截然不同的设备，尽管每个感知的自由度数目相同。Polhemus Fastrak是一个自由空间磁性跟踪设备，曾在前面的一章中做过介绍。Spaceball（空间球）感知施加于安装在固定底座上的球的力和力矩。虽然两个设备感知的自
451 由度数目相同，但它们对于相似任务的使用方法完全不同。Spaceball相对于Polhemus Fastrak的一个优点是它是置于桌面上的，无需手持于空中。然而，Spaceball也有它的难处理之处，那就是很难在施加力矩的时候不产生某个方向的力，反之亦然。因而通常我们在一个时间段中只允许一个旋转或平移，至少对于非专家是这样的。

图表中还显示有VPL数据手套（Zimmerman et al.，1987），它已经用在桌面系统中，虽然它更常用在沉浸式系统里。数据手套感知每个手指相邻两个关节之间的弯曲角度，并可以选择是否感知拇指、食指和中指的伸展。所有这些构成了13个自由度的感知设备。

21.3　选择

选择是一种能力，通过注视、指点或接触一个对象来表示对象成为注意的焦点。我们在

⊖　假设我们使用一个以身体为中心的轴坐标系，X指向身体的右侧，Y指向里，Z指向上。

这里介绍它，是因为在虚拟现实模型中，它是对象操作或移动的一个不可分离的部分。事实上，选择是源于桌面隐喻的一种技术，对象可以通过在它们上面点按来选择，或者用一个橡皮筋型隐喻套住它们来选择。在虚拟现实模型中，采用了现实的隐喻，指示对象没有什么效果。然而，在更奇妙的环境中对象会自己对这些手势产生反应。在桌面模型中，预先选择对象通常是重要的，因为这给交互控制的设计提供了更大的自由度[⊖]。

在虚拟现实模型中，对“指点”的定义依赖于用户的手势是否得到了感知。如果没有手势是有效的，通常的方法是通过光线相交测试找到第一个与光线相交的对象，测试所用的这条光线从用户的手出发，并沿着手的方向。

基于光线的选择隐喻适用于任何系统，无论用户是使用二维光标或是三维光标。举例来说，在一个二维系统中，通过鼠标从虚拟镜头发出一条光线穿过光标所指的那个屏幕像素，我们就可以选择一个对象，这与先前使用光线投射来绘制是同样的方式。可以看出这与虚拟现实模型中的选择有很大不同。在虚拟现实模型中的选择甚至不需要用户看着目标，因为光线是从手而不是从眼睛出发的。图21-2给出了一个比较。

这种基于光线投射的方法优点是对象可以在远处被选择，而在虚拟现实模型中，用户必 452
须首先靠近对象，使之在自己可以触及的范围内选择。缺点是选择对象之后在很远的距离上操作它不是一件容易的事情。首先，由于被选择对象的这种距离使得操作误差得以放大；其次，不可能让对象绕某个轴旋转，除非是绕光线本身旋转。

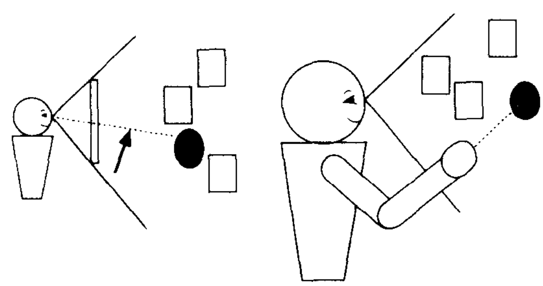


图21-2 比较桌面模型中的选择和虚拟现实模型中的选择

21.4 操作

计算机用户对标准二维桌面上的对象操作技术会感到非常熟悉。在三维中的对象操作比在二维中要困难得多，这有多种原因。不仅仅是有更多的自由度需要控制（六个自由度而不再是两个或三个），而且通常的交互设备不能同时控制所有的这些自由度。二自由度设备使用很广泛，许多方法都是用鼠标和手柄来控制对象，要么直接使用模式切换在多个方向上激活平移和旋转，要么通过虚拟设备间接完成。

使用二维设备的平移

使用二维设备控制对象有很多种可能性。给定输入设备的相对位置，在某个坐标系统中

⊖ 例如，在VRML浏览器检查模式中利用选择来指定对象围绕谁进行旋转，随后鼠标运动移动视点，这样控制方法就有两个可分离的状态。

可以简单地将它映射到三个轴中的两个轴上。问题是有很多可能的坐标系可以选择：世界坐标、所选择对象的局部坐标，或是所选对象任意相关父对象的坐标系（如图21-3）。给定多种可能性，通常我们都在屏幕上画出坐标轴，但是即使如此控制还是很不直观。为了只对应两个维度，应该有一种替代模式，要么在第三维中切换，要么在不同的一对轴之间进行切换。

基于对象坐标的操作的另一种方法是将二维设备的移动映射为相对于照相机坐标空间的平移。这是比较直观的，举例来说，设备的左右移动映射为屏幕上的左右移动。如果二维设备控制屏幕上的光标，通常平移量根据在屏幕内的深度有不同的放大比例，这样外表上的运
453 动保持对象与光标的相对固定。

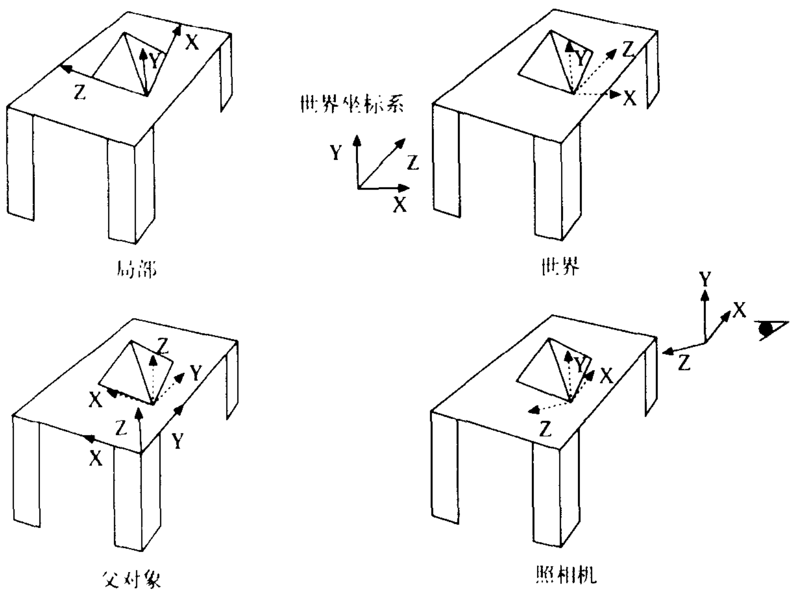


图21-3　在不同的坐标系中控制对象

使用二维设备的旋转

同样对于旋转控制也有许多可能性。鼠标的移动可以映射为关于任何两个轴的旋转，坐标轴的选择也有相当多的组合。

以视点为中心的控制技术同样是有用的。Chen等（1988）给出了一个虚拟球技术，对象被看作包含在一个球里面。在球面上的二维设备运动使得球体绕着镜头的U轴和V轴滚动，在球体外面的二维设备运动转变为绕N轴的旋转。对球体拂掠一遍将旋转180度，而绕球体一周表示绕 N轴 360 度旋转。

21.5　移动

控制视点也是一个困难的任务，它包含与对象操作一样的自由度数目。我们在“以身体为中心的导航”中讨论过虚拟现实模型中的移动。对于桌面模型，移动控制是非常不同的，它如同对象操作，我们必须将控制从二维设备映射到六维任务。

存在两种基本方法，一种是在工作空间中移动视点（这就是我们所指的移动），或者围绕

视点移动工作空间。本质上二者的差别在于执行平移和旋转所选择的坐标系不同。Ware和Osborne（1990）给出了三个导航隐喻来说明这种差别。

掌握场景。场景本身完全服从于输入设备。 454

掌握眼球。输入设备的移动直接对应于眼睛的移动。

飞行媒体控制。输入设备提供对速度和旋转等媒体工具的控制。

操作掌握场景的隐喻等同于对象操作隐喻。它很适合于对小场景的观察，此时用户的视线总是围绕着某个指定对象转。它除了提供旋转功能外还提供缩放功能。VRML浏览器的“检查模式”就是这种隐喻的一个很好的例子。当场景变得很大、眼睛需要遍历内部空间的时候，这种隐喻就很麻烦了。

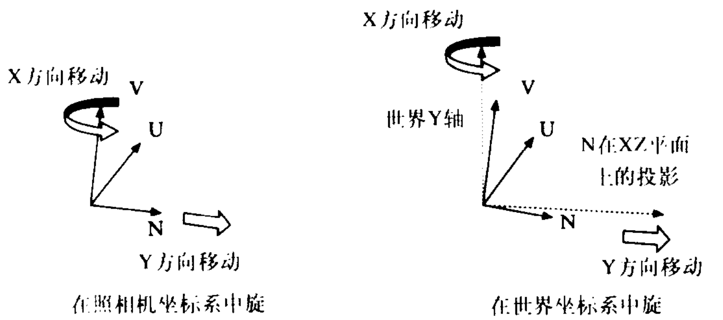
掌握眼球和对飞行媒体工具控制之间的差别在于输入设备是直接映射为定位和旋转，还是映射为视点的速度。有一系列可行的技术，我们将在下一小节中讨论其中几个例子。

用二维设备指定移动

对于视点的控制，以视点坐标为中心的技术显然是最恰当的方式。然而，平移和旋转通道的分离是多种多样的。

对自由导航通常采用的一种技术是通过二维设备的前向和后向移动来实现视点的移动，这是沿着视点坐标系负N轴的移动，并让左右移动转变为绕V轴的旋转。

对于导航在平面上（例如漫游仿真）的绝大多数情形，通常会有两种变化。第一是照相机围绕世界坐标的Y轴，而不是V轴。请注意选择Y和选择V的不同。前者是在世界坐标的XZ平面中旋转照相机，后者是在照相机的UN平面上旋转。第二是视点的平移通过将VPN投影到世界坐标的XZ平面来实现。两种技术之间的差异由图21-4说明。



在照相机坐标系中旋转、没有平面约束　　在世界坐标系中旋转、有平面约束 455

图21-4 用二维设备实现移动

如同对象操作隐喻，需要一系列的模式来产生绕剩余轴的旋转和沿着该轴的平移。另一个常用的技术是让鼠标在一般情况下控制前后倾斜和左右偏转（关于U和V的旋转），第二个模式是沿着U和N的平移，第三种模式是围绕N的旋转和沿着V的平移。

范围和精度

对于长距离的位置控制是精确的，但是效率比较低。速度控制允许在很大的距离上迅速移动（Ware and Slipp，1991），但是当接近一个对象的时候精度不高。我们可以在两个控制

方法之间转换，或者是根据当前的工作范围进行强制转换。

在对数逼近技术的应用中发现一种更一般的办法（Mackinlay et al.，1990）。在该技术中，用户通过对象的选择指定兴趣点。之后，用户的视点平滑地沿着一条路径移动，该路径朝向那个兴趣点，由对数逼近产生。到兴趣点的距离可以通过下列等式计算出来，这里d是初始的距离，k是一个常数，可以用来控制逼近的速度。

$$f(t) = d - de^{kt} \tag{21-1}$$

这通常是结合掌握场景隐喻一起使用的，选择点就变成旋转的中心。

在对象是关注焦点的某些情形中，一种适用的技术是自动照相机控制技术（Phillips et al.，1992）。在此技术中，照相机自动寻找一个好的位置以便执行预期的任务。自动照相机摆放的主要问题是选择一个观察角度，使得通过这一点看去，焦点对象不会受到遮挡。这可以使用半立方体方法来计算。场景被投影到围绕焦点对象的半立方体上。半立方体的未遮盖区域就是照相机可以摆放的位置。如果场景包围了焦点对象，那么对象在半立方体上的投影深度就必须考虑到。

虚拟人体的使用

我们已经看到虚拟人体对于沉浸式环境中是有用的，但是它对于桌面情况的意义就不是
456 那么明显了。通常虚拟人体的本质是世界中的一个三维光标，如果使用的是鼠标这样的二维定位设备，那么连三维光标也是多余的。

使用自我为中心的人体是受限制的，但是对于漫游系统，尤其是三维游戏，通常使用一个外在的人体，此时视点位于被渲染的化身的后上方，该化身与环境进行交互。然后用户控制映射到化身的平移和旋转上，视点相对化身保持固定。遗憾的是，在封闭环境中化身很容易受到遮挡，所以就必须有某些形式的自动镜头控制方法来找到合适的位置。很多时候可以通过将照相机拉近来解决，但是通常这样做会失败，照相机需要从它现在的位置移动到化身的后面。彩图A-1给出了一个VRML浏览器中从化身肩膀上看的视图。

21.6　界面中通道的屏蔽

在虚拟现实模型中，由于视点的控制和对象的操作是通过不同的设备（跟踪器）来执行的，这些设备可以同时操纵，所以就需要屏蔽某些通道。遗憾的是，对于桌面设备的绝大多数组合是没有这些构造的，所以我们就需要做出一些让步。

对于对象操作，如果用户确实需要精细控制，就很难降低这种复杂性。Nielsen和Olsen（1986）给出了一种能减轻负担的技术。他们将二维移动映射为沿着虚拟世界坐标轴的移动，该坐标轴的选取要使得鼠标的移动方向与该轴有最靠近的二维投影。这样二维设备的移动平面就被分割为六个区域，分别对应于各个轴的正负方向，如图21-5所示。如果所选取对象的局部坐标投影如左图所示，鼠标在A区间的移动映射为沿着局部X轴正方向的移动，在B区间的移动映射为沿局部X轴负方向的移动，在C区间中的移动映射为沿局部

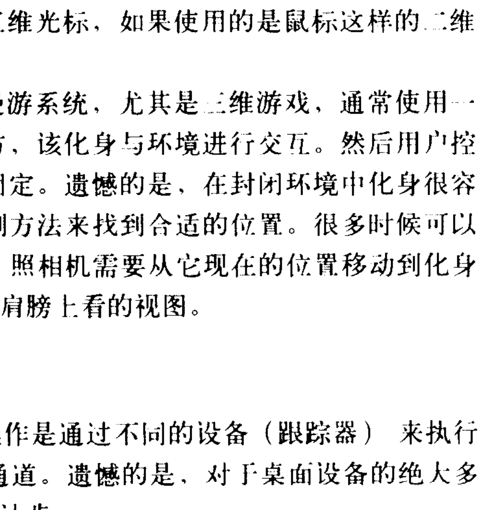


图21-5　根据对象轴投影之间的相互关系将二维移动映射到三维

Y轴正方向上的移动，等等。

约束

无论是对象操作还是视点控制，降低复杂性的一种方法是减少可控制自由度的数目。举例来说，一个位于桌面上的对象的运动就是被约束为在桌面这个平面上的平移运动，而不能 457
有垂直于桌面平面的运动。当运动由鼠标控制的时候，通常没有提供转动照相机的机制，除非采用了特殊的模式（例如按其中一个Ctrl 键）。这样做对于漫游系统是有用的，甚至是非常合适的，但是对于一般对象的观察就是一个障碍了。

在二维中约束的使用是相当普遍的，例如基于网格方法或重力吸引的方法。存在很多类似的3D技术，例子可见（Bier, 1990）。

虚拟小构件

虚拟设备的使用已经在前面有关连接组成那一部分讨论过。我们可以对此进一步深入分析，介绍允许用户借助代表某种操作的 3D 对象来控制对象的操作。彩图21-6给出了这样一个例子。

Open Inventor操纵器如彩图21-6所示，它由一系列允许对对象实施各种变换的虚拟小构件（Wernecke，1994）组成。存在很多模式，每一种都可以选择和抓取第一幅图中三个可见元素当中的一个：线框立方体表面、线框立方体立方柄、位于对象中心轴上的球形柄。第一个的效果是在由该面所定义的平面上平移（彩图21-6，右上），第二个的效果是缩放（左下），第三个的效果是旋转。旋转发生在两个阶段，首先（中下），我们看见旋转控制的创建，但只允许绕两个已指示的轴中的一个旋转。当移动鼠标进入某个已指示方向的特定距离内的时候，我们就将旋转约束为在该方向上的旋转（右下）。在文献中（例如 Brookshire Conner et al., 1992）可以看到很多这种控制的例子。

21.7 C语言例子

3D游戏的交互方法是让镜头的旋转与鼠标相连，让移动与光标键相连。GLUT库有一系列辅助函数，允许程序设计者非常快地建立简单的界面。特别地，它允许键的按下、键的释放、鼠标事件的回调注册，并提供了一个非常简单的菜单系统。下面的代码片段表明了对鼠标移动事件的注册、一个简单的菜单系统和按键事件： 458

```
glutIdleFunc(idle);
glutMouseFunc(mouseButton);
glutMotionFunc(mouseMotion);
glutKeyboardFunc(keyboard);
glutSpecialFunc(specialDown);
glutSpecialUpFunc(specialUp);
glutCreateMenu(menu);
glutAddMenuEntry("Toggle planar constraint", M_PLANAR);
glutAddMenuEntry("Move Faster", M_FASTER);
glutAddMenuEntry("Move Slower", M_SLOWER);
glutAddMenuEntry("Rotate Faster", M_RFASTER);
glutAddMenuEntry("Rotate Slower", M_RSLOWER);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

glutIdleFunc的作用将在下面讨论。注意这里有两个键按下的函数类型：glutKeyboardFunc

是生成ASCII代码的键，glutSpecialFunc是其他的一些键，如光标键和功能键。在GLUT3.7中，还增加了函数glutKeyboardUpFunc和glutSpecialUpFunc，它们提供键释放事件的通知。

我们创建一个简单的菜单系统以便允许用户改变平移和旋转的速度，并提供平面约束的打开和关闭。下面的代码段说明了这些是如何实现的：

```
enum {
   M_PLANAR,
   M_FASTER,
   M_SLOWER,
   M_RFASTER,
   M_RSLOWER
};

int usePlanarConstraint = 1;
double velocity = 0.05;
double angularVel = 0.005;

void menu(int item)
{
   switch (item) {

   case M_PLANAR:
      usePlanarConstraint = 1 - usePlanarConstraint;
      break;
   case M_FASTER:
      velocity*=1.5;
      break;
   case M_SLOWER:
      velocity*=0.5;
      break;
   case M_RFASTER:
      angularVel*=1.5;
      break;
   case M_RSLOWER:
      angularVel*=0.5;
      break;
   }
   glutPostRedisplay();
}
```

459

实际的平移运动与光标键相连：

```
static short Move = 0;
static void specialDown(int key, int x, int y)
{
   switch (key){
   case GLUT_KEY_UP:
      Move = 1;
      break;
   case GLUT_KEY_DOWN:
      Move = -1;
      break;

   }
}


static void specialUp(int key, int x, int y)
```

```
{
  switch (key) {
  case GLUT_KEY_UP:
    Move = 0;
    break;
  case GLUT_KEY_DOWN:
    Move = 0;
    break;

  }
}
```

一般在全屏幕游戏中，鼠标的移动永远是与照相机的旋转联系在一起的。此时我们是在窗口环境里面，因此通过按下鼠标左键来表示激活照相机旋转：

```
static GLint XC, YC;/*current mouse position*/
short Rotate = 0;

static void mouseButton(int button, int state, int x, int y)
{
  if(button==GLUT_LEFT_BUTTON){
    XC = x;
    YC = Height - y;
    if(state==GLUT_DOWN) Rotate = 1;
    else Rotate = 0;
  }
}
```

460

如果鼠标移动，我们就施加旋转和可能的平移到照相机：

```
static void mouseMotion(int x, int y)
{

  if(Rotate){/*for left button*/
    rotateVPN(x,y);
  }

  if(Move!=0){/*for right button*/
    moveVRP();
  }

  clickView_GL(TheCamera);

  /*force a call to display*/
  glutPostRedisplay();
}
```

我们也使用idle函数周期性施加平移变换，因为只得到键按下和键释放事件，而不是在光标键保持按下时得到事件：

```
static void idle(void)
/*if nothing else happening*/
{
  if(Move!=0) {
    moveVRP();
    clickView_GL(TheCamera);

    /*force a call to display*/
    glutPostRedisplay();
  }
}
```

最后我们提供两个函数，即moveVRP（照相机平移）和rotateVPN（照相机旋转）：

461
```
static void moveVRP(void)
/*move the VRP along the VPN*/
{
  Vector3D n;
  Point3D vrp;

  vrp = TheCamera->vrp;

  /*normalize the vpn - pity this happens also in camera code*/
  normalizeVector3D(&TheCamera->vpn,&n);
  vrp.x += (velocity*n.x)*Move;
  vrp.y += (velocity*n.y)*Move;
  if (!usePlanarConstraint) {
    vrp.z += (velocity*n.z)*Move;
  }

  setVRP_GL(TheCamera,vrp.x,vrp.y,vrp.z);
}

static void rotateVPN(int x, int y)
{
  double dx, dy; /*change in mouse position*/
  double nx,ny,nz; /*new vpn*/
  RotationMatrix r;
  /*trial and error value affecting angular velocity*/

  /*difference between old and new coordinates*/
  dx = angularVel*(x - XC);
  dy = angularVel*((Height-y)- YC);

  XC = x;
  YC = Height-y;

  /*treat (dx,dy,1) as the new offset VPN, expressed in VC*/
  /*transform back to WC, using transpose of RotationMatrix*/
  r = TheCamera->R;
  nx = r.m[0][0]*dx + r.m[0][1]*dy + r.m[0][2];
  ny = r.m[1][0]*dx + r.m[1][1]*dy + r.m[1][2];
  nz = r.m[2][0]*dx + r.m[2][1]*dy + r.m[2][2];

  setVPN_GL(TheCamera,nx,ny,nz);
}
```

21.8 VRML例子

VRML不但描述动画和脚本，而且提供多种交互技术。有两类基本的交互：在场景中视点的用户控制、用户和场景中对象的交互。它们的描述方式非常不同。视点控制是通过世界的创建者从多种预先规定的媒体工具当中选择一个，而且在同一时间只有一种媒体工具处于激活状态。相
462 反，与对象的交互是通过场景图中传感器节点检测与该传感器节点下面的几何节点之间的交互。

媒体工具

VRML本身并没有明显给出如何进行世界导航。然而，它给出了两个浏览器提供的常用交互隐喻。它们是行走媒体和检查媒体。行走媒体工具类似于一种简单的飞行媒体工具。用户假

定是在一个水平面上操纵虚拟世界，鼠标控制映射为左右摇摆和前后的运动。在检查媒体工具中，导航围绕一个焦点，鼠标控制映射为围绕该焦点的旋转。这与掌握场景隐喻很类似。

媒体工具也有关于场景中用户化身表现的一些非常基本的东西。这个化身不是只为渲染的，而是用做用户和场景之间的碰撞检测。场景中的元素可以制造成实心的，用户不能够穿过它们。还可以产生重力，这样用户在世界环境中的运动就会保持在表面上。化身是定义在NavigationInfo节点里的。它包含一个域叫做avatarSize，其类型是MFFloat，默认值为[0.25，1.6，0.75]。前两个值给出简单柱状化身的半径和高度，该圆柱体不会穿透到任何实体对象中。第三个值规定了迈步的高度。如果圆柱体碰到场景中任何低于此高度的几何实体，它就移动到该几何实体的上面。这允许化身沿着表面小步前进。

传感器

VRML允许对象编程来响应用户的指点、选择或操作部分场景图。这是通过将场景图中节点配上一个传感器节点来实现的。当传感器被附在场景图的一个分支的时候，每当用户使用指点设备与该分支下面的任何几何实体进行交互时，传感器就能够接收某些输入事件。传感器被激活的实际细节依赖于所使用的实际设备，我们这里只给出用鼠标进行交互时的激活过程。

TouchSensor对用户把鼠标光标放于几何实体上这个行为做出响应。一旦鼠标光标落在几何实体上，当用户按下鼠标键的时候它就产生第二个响应。PlaneSensor同样对用户将 463
鼠标光标放于几何体上做出响应，但是如果用户按下鼠标键并保持着，他或她的拖动运动映射为平面上的移动，而且该平面移动将作为传感器的SFVec3f eventOut。这个矢量可以用于驱动一个Transform节点，这样几何体本身就可以被来回拖动了。

其他的交互传感器包括：SphereSensor、CylinderSensor、Anchor以及Collision。SphereSensor和CylinderSensor与PlaneSensor类似，但是鼠标移动分别映射为三维和一维的旋转。Anchor节点是HTML超链接的三维等价物，也就是说，它能使新的世界和其他类型媒体文件载入到浏览器的一帧中。Collision可以用来检测何时用户化身会碰到一个场景实体。

VRML规范故意模糊传感器如何被激活，因为这依赖于输入设备，而且需要浏览器提供一些适当的机制。对于二维桌面的一般情况，传感器的激活是通过鼠标光标在对象上的移动，用户对准对象点击鼠标，这是一种基于光线的选择。当VRML在沉浸式系统中使用时，可以用传感器激活隐喻替换，只要用户触摸它们（Stiles et al.，1997）。

21.9 小结

在这一章中我们介绍了桌面虚拟现实系统的交互问题。可以看到对于这类系统存在着大量的交互设备，也存在着各种各样可用的交互技术。不像二维界面系统，这里几乎没有任何标准，三维应用的交互方法也是不同的，有时这种不同又相当细微。

来看一下三个会用到的基本交互任务：

- 选择——指示屏幕上一对象为后续行为的焦点。
- 操作——移动或改变一个对象。
- 移动——在环境中移动一个指定对象或照相机视点。 464

假设每个任务都包括六个自由度，我们也介绍了能降低或屏蔽界面中通道的一些技术。特别还介绍了小构件界面。

第六部分 从真实到实时 II

第22章 基于光线的全局光照方法

22.1 引言

本章简要介绍基于光线的多种全局光照方法。这些技术有很多根本性的不同，最有效的是应用光亮度方程的随机（蒙特卡洛）解。所介绍的一些技术在前面的章节中已经研究过了，但是这里我们再总体回顾一下。第3章的光亮度方程也要再进一步研究和扩展。

22.2 光线跟踪方法

光线跟踪

光线跟踪是由Whitted（1980）引入的。光线跟踪需要针孔照相机模型和点光源，以及非参与性媒体（事实上是对空气近似表示的真空）。它的基本操作是跟踪一条始于投影中心（COP）并经过图像平面某个点的光线。点是一个像素（或“子像素”，如果在需要使用一些反走样方法的时候）——目标是求出像素的“亮度”，方法是反向跟踪光线到场景，从一个对 465
象反射到另一个对象，直到它离开场景或对亮度的贡献可以忽略不计为止。

为了达到这个目的，我们求出与光线相交的最近的那个对象（在点p）。双向反射（BRDF）函数对于最一般的情况将会确定光线的后续路径。事实上，光线跟踪在光线与每个对象相交点上只有三个特殊路径。第一个是“阴影感知器”光线——光线从对象相交点p到各个点光源，如果每条这种光线在到达光源之前不与另外的（不透明物）对象相交，则它为这一点增加一点局部光照贡献。亮度是三项之和，代表场景中所有背景光的“环境”项、对于理想漫反射器每个光源基于朗伯定律的项，以及基于Phong模型的加亮显示项。值得注意的是，这里对于漫反射没有全局光照，而只有每个光源对p（因而到当前像素）的局部贡献。如果对象是理想的镜面反射器，那么计算出镜面反射方向上的一条新的光线（根据入射角等于反射角，且反射光线和入射光线在相同的平面中这样一条定律）。光线跟踪函数使用反射光线递归调用，其结果添加到点p的亮度。同样地，如果对象是透明的，那么使用 Snell定律，穿过对象射出去的新光线方向可以计算出来，光线跟踪函数被再一次递归调用来产生一个亮度并增加到p点的亮度上。这个光线跟踪函数被每一条由COP点出发经过每个图像平面上相关点（像素或子像素）的主光线调用一次，结果是得到一幅模拟镜面反射和传导的图像。

光线跟踪是计算机图形学中具有照片真实感图像合成的一个主要进展。它的计算量非常

大——大部分计算量主要是在对光线-对象的相交计算上。一个质朴的实现会需要对每条光线和每个对象做测试。在光线跟踪上的大量研究都是集中在通过充分减少光线-对象相交测试的数量来加速这个过程。如我们在第16章中所看到的，这些方法包括包围体和层次结构（也可
466 参见 Clark，1976；Rohlf and Helman，1994），这里光线首先对封装实际场景对象的简单对象进行测试，只有当对简单包围体对象（这是一个轴向对齐的方盒或球体）测试成功，才有必要进一步测试光线与实际场景对象的交。这种包围体可以根据场景中对象之间的空间关联关系组织成层次结构——这样，一组靠得很近的对象可以集中在一个包围体内，然后再将包围体归类到更大的包围体中，等等。包围体可能被组织成一棵树，树的根节点封装了整个场景以及位于树叶上的对象。光线沿着树向下过滤，直到没有与之相交的包围体，否则最后在树叶处与一个对象相交。

空间分割方法无遗漏地将场景所占据的空间分割成一些较小的子空间，每一个这样的子空间维护一个对象识别符列表，表中对象都位于该子空间中。一致分割法（Fujimoto et al.，1986；Cleary and Wyvill，1988）把场景空间分为规则的小三维立方体或单元网格。穿越这个空间细分的光线路径可以很快计算出来，只有在光线路径单元中的对象才是相交的候选对象。自适应分割，例如八叉树（Glassner，1984），对空间的分割依赖于对象的分布——越是对象分布密集的区域被细分的程度就越大。虽然有大量的研究，光线跟踪还是一个相当费时的计算过程。显然视点每变化一次，整个光线跟踪算法就必须重新执行一次。已经有一些努力在研究当照相机移动时从图像到图像之间的关联性（例如Chapman，1991；Teller et al.，1996），但是还没有取得实时漫游可用的充分结果。

光线跟踪的光线空间方法

光线分类模式提供另一种加速光线跟踪的方法。主空间此时是光线空间，而非通常的对象空间。对此最著名的例子是由Arvo和Kirk（1987）给出的，在第16章中已讨论过。一条光线有一个原点（x，y，z）和由角度（θ，ϕ）给出的方向，因此可以看成是5D空间中的一个点。Arvo 和Kirk事实上将场景中所有可能的光线表示为六个5D点的集合。用一个轴向对齐的包围盒将场景包围住，包围盒的每个面有一个二维UV坐标系。因此一条光线可以通过它的原点（x，y，z）、与它相交的包围盒的一侧（左、右、上、下、前、后）以及相交点的UV坐标来表示。因此所有可能的光线可以由六个（x，y，z，U，V）点的集合来表示。

算法利用光线关联性——即彼此“很近”的光线最有可能与相似的一组对象集合相交。简要地说，每条光线被看作32叉树细分（八叉树的5D版本）分别对六个面自然形成，这里在任何时候，树的任何叶节点表示一条（相似）光线，有其对应的“候选”对象集合。也就是说，在候选集合中的任何对象都有可能至少与对应光线子集中的一条光线相交。

很重要的一点是，“所有可能光线”的空间比对象空间更为重要。算法构造光线空间的分割而不是对象空间的分割。

Muller和Winckler（1992）介绍了另外一个光线-空间方法——尽管有相当不同的理由。“广度优先光线跟踪”的设计不是专门为提高速度，而是为有大型对象数据库时节省实际内存。
467 其思想是光线保持在主内存中，对象信息可能在磁盘上储存。每条光线储存成一个原点、表示光线直线方程的参数、当前相交点、沿着光线的当前相交点，以及对应于那个相交点的对象。该算法的一个质朴实现中，每个对象从磁盘读到内存并与所有的主光线进行相交测试。

使用一种"z缓冲区"技术，所有与任何对象相交的主光线都指向与之相交的最近的那个对象，当然，还有对应的相交点。对所有的"阴影感知器"光线我们重复相似的过程。然后是对所有的反射光线、传输光线等等，到任何必须的深度层次。因而光线跟踪是广度优先的——在每个反复中处理所有的光线而不是递归地在对象空间中移动。Nakamaru 和 Ohno（1997）后来通过对光线使用一致的空间细分改进了这个方法。事实上这个空间细分是在对象空间中执行的——对象空间被分割成规则的立方体单元网格，然后每个单元维持一个穿过它的光线的指针列表。通过这样的方式对象从数据库中读出来，只有那些穿过存在对象单元的光线需要测试。虽然广度优先的光线跟踪避免了递归，可以处理极为庞大的数据库，但是它通常比正常的深度优先光线跟踪要慢，当然它也具有所有一般光线跟踪的性质——在全局层次上处理镜面反射，且具有视点依赖性。

离散光线跟踪

离散光线跟踪（DRT）属于计算机图形学中的体可视化方法（Kaufman，1996）。它是一种不同于光线跟踪的方法，是由Yagel等（1992）引进的。在传统的计算机图形学中基本的实体是"像素"，表示显示设备上的一个位置，它可以独立地设定为不同的颜色强度。在体渲染中显示空间是三维的，最小的对应实体是体素，它可能既包含颜色信息又包含关于那个所属（立体）对象的信息。实际上，体素是很小的，是立方形的非重叠的区域，小到最多只有一个对象与之相交。

在DRT中对象首先"渲染"到体素空间。每个相交体素记录关于初始对象精确法线的信息、关于对象材质属性的信息，也记录所跟踪的光线信息，对整个体素空间和每一个光源。每个光源体素记录一个两比特的代码，指示它是可见、不可见的，还是经过另外一个半透明对象可见。 468

经过体素空间的光线跟踪包括求出沿着光线路径的"最佳"体素集合，这很像Bresenham 算法或 DDA 算法求出在二维像素空间中最佳的直线表示。

渲染阶段的开始如同传统光线跟踪中那样。光线的跟踪始于COP原点并穿过二维图像平面。光线沿着体素路径前进，直到遇到一个非空的体素。当然，它找到的第一个体素就是相应的交点。"阴影感知器"光线我们已经知道了（它们在对象渲染阶段之后立即得到预计算）。接下来DRT遵循了标准的递归光线跟踪方法——衍生反射和传导光线。

与传统的光线跟踪相比，DRT 的优势是：

- 相交计算不再是主要的了——光线前进直到遇到第一个非空体素。无需搜索最近的相交点，但是花费在光线穿越体素空间上的时间却变得很可观了。
- 光线跟踪时间本质上与对象的数目无关。正如作者所指出的那样，对象数目越多，实际上花费在光线跟踪上的时间反而越少，因为光线在对象分布比较浓密的空间区域中遇上非空体素的时间越短。当然，光线跟踪时间确实依赖于体素空间的3D分辨率，因为它确定了光线遍历的路径长度。
- 许多视图独立属性预先计算——例如阴影感知器、法线和纹理。因此相对于传统光线跟踪，视图的变化所需要的重新计算大大减少。

DRT显然是高度视图依赖的方法——绝大多数计算（为每一个图像平面的像素所构造的光线跟踪树）需要因视图的改变而重新执行。

22.3 分布式光线跟踪

蒙特卡洛方法

蒙特卡洛是一种一般的基于统计采样的估计方法。最著名的例子是对π的估计。在一个边长为2的正方形中嵌入一个单位圆。现在随机地选择大量位于正方形中的随机点（例如通过往正方形里投掷大头针）。正方形区域面积是4，圆区域的面积是π，因此位于圆圈中的大头针比例估计为π/4。统计中的大数定律保证对于概率1的任何误差范围都可以经过充分大的样本
469 量来达到。现在同类技术可以用于求解光亮度方程。不是经过像素发射一条单一光线，并从每个表面沿着几何计算的方向反射出去（反射或折射），而是采用随机采样模式发送出多条光线，然后将光线的结果汇总在一起。有许多不同的方法来做这件事。我们首先考虑分布式光线跟踪，其次考虑路径跟踪。

分布式光线跟踪（Cook，1984）极大地扩展了由经典光线跟踪所能获得的结果，它的实现是经过每个像素跟踪多条主光线并且在每个后续步骤中跟踪多条光线。尤其分布式光线跟踪能对平滑表面建模，而不是仅仅能反映镜面反射；能对半透明建模，而不仅仅能对强透明性建模；同时它还能对阴影半影建模，而不是只对本影建模，而且，采用经过透镜跟踪而不是使用针孔照相机得到景深效果，在一段时间上采样光线来形成运动模糊。它也是克服光线跟踪图像中走样的一种方法，不仅仅是通过超采样而是通过在一个特殊类型的随机模式下对像素区域上光线进行分布的方法。这降低了走样并通过最后图像中可容忍的噪声来代替它。

走样

图22-1显示了源自COP并穿过视图平面像素的一组主光线。对场景有两个部分分别标记为A和B。包含场景的“信号”是连续的。显然我们采样场景是基于经过像素位置的离散的一组光线。通过像素采样容易看出理论上可能精确再生出标记为A的场景部分，然而不可能再生
470 出标记为B的场景部分。通过对比采样速率和两个信号的频率，我们能清楚地看出其差别。对于信号A，信号中的频率（在我们的二维模拟信号中两像素位置所构成的边界内的变化数）少于一个周期。对于信号B，在两个连续的像素采样位置之间的空间中周期数为2或更多。Nyquist（奈奎斯特）限制允许在两个像素之间的最大值为一个周期，以便信号可以再生。如果超过该界限的话，信号就无法从采样模式中重建。当渲染低采样信号（如B）时，它将会形成走样，即在图像重建中出现最初信号没有的效果。

减少这种走样影响的一种方法是提高采样频率。在光线跟踪上下文中，这通常叫做超采样。它的意思是每个像素一致地分割成子像素，一般的光线跟踪是在子像素层次上执行的。当一个像素里的所有子像素结果都计算出来之后，对这些结果求平均以便产生最后的像素亮度。这种超采样能减少走样，但是还仍然会有信号打破采样限制且导致走样。

在分布式光线跟踪中可采用随机采样模式。一种可能是在像素区域上一致且随机地分布样本点。另一种在样本估计中产生比较小方差的模式是分层采样。举例来说，要估计居住在某个城市中的成年人的年平均收入。一种技术是获得所有居住在该城市里的人的列表（这种列表来自于选举记录），然后从这个列表中随机选择n个人，通过采访了解他们的年收入。假设整个人口的某些特征是预先已知的：例如男女性的比例、年龄构成、职业分布等等。随机

采样越是准确反映出这些大众特性、样本估计的方差就会越小（也就是说准确度就可能比较高）。分层采样就是在各个阶层内随机地采样，所以在样本中各个阶层的比例与人口中比例相一致。在最简单的例子中，如果决定只根据性别分层，男性在总人口中的比例是54%，那么如果样本大小为1000，其中540会是随机选择出的男性，460会是女性。

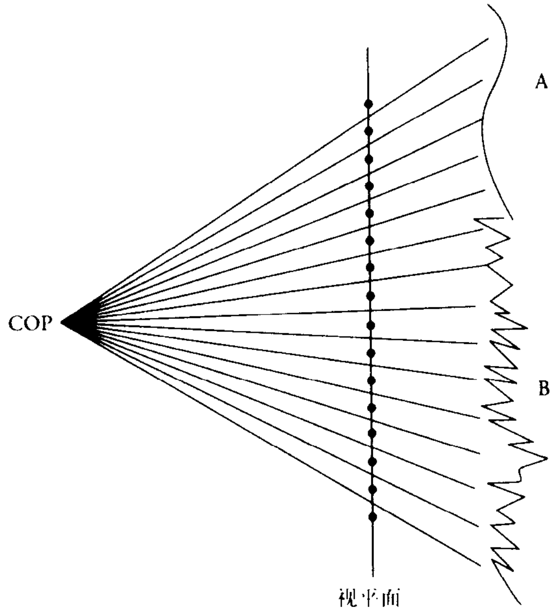


图22-1 由像素采样所导致的走样

现在在一个像素位置（想像成一个正方形）上的随机点的集合不意味着点会均匀散布在像素上。随机绝不意味着均匀散布（玩彩票的人都知道这一点）。随机地选择其要点就是“随机”。在从1到40中抽取6个数的彩票玩法中，结果1、2、3、4、5和6与其他数有完全均等的选择机会。通常用于采样像素的分层采样模式称为抖动采样，如图22-2所示。这里像素被分割成16个子像素。在每个子像素里面选择一个随机位置。因此我们保证适当的区域覆盖（这是分层化），而在每个区域中又是随机的。每个像素在这种抖动方式中采样，光线从子像素内所选择的位置处发出。像素的最后结果（在最简单的情况下）为所有该像素内子像素的结果的平均。 471

这种抖动采样模式能十分有效地降低走样的程度，因为现在我们可能使用这个模式对一个信号周期中任何部分进行采样了。

反射和光照模型

在光线跟踪中当一条光线打在某个物体表面上时，可能会衍生出两条新的光线：一条沿着反射的方向；如果表面是透明的话，另一条沿着折射的方向。因此，所有反射和透明性都是很锐利的。分布式光线跟踪考虑到平滑而并非锐利的反射，以及半透明的而并非很强的透明性。首先考虑反射。对于一个理想的镜面表面有一个反射方向。一个平滑的表面在这个镜面反射光线周围有一组反射光线，这组反射光线存在一个分布（如图6-6所示）。为了要模拟它，我们可以围绕理想的镜面反射方向建立一个抖动采样模式（见图22-3）。

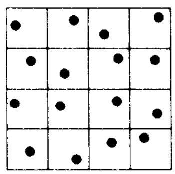


图22-2 在像素上的抖动采样

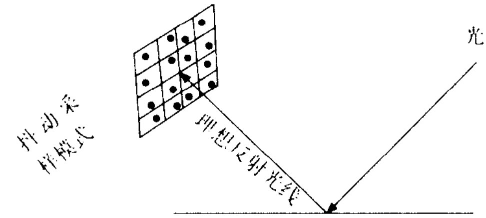


图22-3 光滑反射的抖动采样模式

472 当一条光线撞击到表面时，就在抖动随机采样方向中生成一条样本光线，而不是沿着理想的镜面反射路径。指派到不同方向的可能性应该由表面的BRDF所决定。在这个交点上的最后结果可以看成是个别光线结果的平均。

对于透明表面完全可以使用相同的过程。许多真实表面不是纯粹的透明表面，而是看起来有些朦胧。如果只使用计算出来的理想折射光线，那么只能得到强透明性效果。然而，抖动采样模式同样可以在理想的折射方向上构造，就像在镜面方向上构造一样，以便获得半透明的效果。

经典的光线跟踪使用点光源。当一条光线撞击到表面的时候，"阴影感知器"光线就被追踪到每个光源点（或方向）上，它要么被另外一个对象（在阴影中）遮挡，要么不被遮挡（它对于光源是可见的）。这种二元选择导致的结果是经典的光线跟踪只能模拟阴影本影。然而，真实光源是有面积的，正如我们已经看到的（参见第14章），这些光源能引起阴影半影——表面上的一点可能对于光源的一部分是可见的。分布式光线跟踪在这方面的处理方式非常类似于反射和折射光线的方式。为面光源建立抖动的采样模式。因此来自每个表面交点的阴影感知器光线不是沿着预先确定的路径发送到光源点上，而是对面光源进行采样，这样任何光线可能性都基于光源上的光亮度分布。

景深

计算机图形学（包括经典的光线跟踪）基本上全是采用针孔照相机模型（参见图5-4）。分布式光线跟踪能通过一个透镜系统模拟视图，因此可以得到如景深这种通常在真实电影上可以看到的效果。对于针孔照相机模型整个图像都位于焦点上，而经过透镜系统形成的图像一部分在焦点上，其他部分则不在。首先，我们简要地讨论薄凸透镜的几何光学（Jenkins and White，1981），然后说明如何将它用于分布式光线跟踪情形中。

图22-4给出薄凸透镜的一个示意图。透镜的轴穿过它的中心，与透镜平面正交。为了便于计算，我们可以将透镜视为平面。有两个重要的点，分别称为主焦点和第二焦点，它们位于距中心点等距离的轴上（哪一个视为主焦点哪一个视为第二焦点依赖于光的入射方向——
473 主焦点与场景位于透镜的同侧，而第二焦点和所生成的图像位于另一侧。）主焦点是这样的一个点，从这一点发出的所有光线经过透镜后的折射方向平行于透镜的轴。所有与透镜平面垂直、与轴平行的平行光线，经过折射后经过第二焦点。（要强调的一点是，两者是完全对称的。光的方向可能颠倒过来，但有相同的结果——此时主焦点变成了第二焦点，反之亦然。）焦点平面与透镜平面平行。

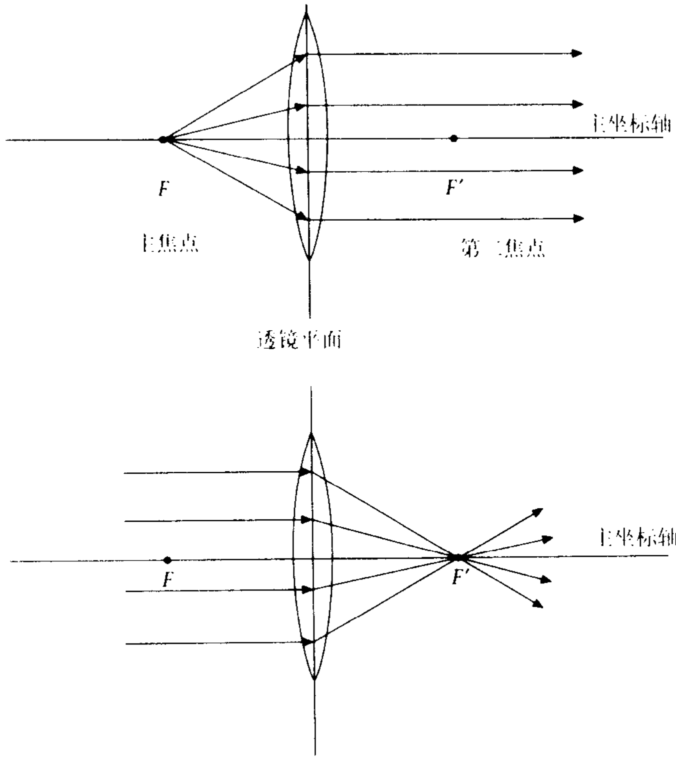


图22-4 薄凸透镜

图22-5给出了图像点如何由一个对象点（在场景中）形成。 474

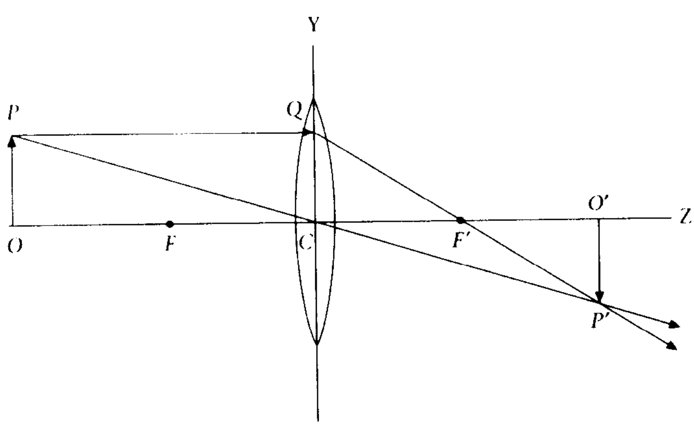


图22-5 图像点的计算

需要注意的是任何经过透镜中心（C）的光线不发生弯曲，而是以直线的方式穿过C。所以给定在场景中的一个对象点P，我们能形成平行于轴的光线，这将会经过第二焦点折射出去。同样也能形成经过P和中心C的光线，图像点就是这两条光线的交点。如果图像平面（平行于透镜平面）位于那个距离的位置上，那么点P的图像正好聚焦在图像平面上（在P'点）。每条从P点出发的光线经过透镜将会相交于P'，换句话说，透镜将P点聚焦在P'处。（严格来说，

只有那些几乎与主轴平行的光线才是这样——也称为轴旁光线。)

假设我们是在左手法则的观察坐标系中，C点在这个坐标系的原点处。假如P在（x，y，-a）（a>0），焦点的距离是f，第二焦点F′在（0，0，f）。光线PQ将交于透镜平面的（x，y，0）。光线QF′有参数化方程：

$$(x-tx, y-ty, tf) \tag{22-1}$$

假设有一个图像平面在z=d处，那么QF与图像平面的交点是：

$$t=\frac{d}{f} \tag{22-2}$$

即在点：

$$\left(x\left(1-\frac{d}{f}\right), y\left(1-\frac{d}{f}\right), d\right) \tag{22-3}$$

同样地，光线PC的方程是：

$$(x-tx, y-ty, -a+ta) \tag{22-4}$$

将与图像平面相交于：

$$\left(x\left(-\frac{d}{a}\right), y\left(-\frac{d}{a}\right), d\right) \tag{22-5}$$

对于P的图像点，若在焦点上，式（22-3）和式（22-4）一定是相同的点。令它们相等，便有：

$$1-\frac{d}{f}=\left(-\frac{d}{a}\right)$$

475

$$d=\frac{fa}{a-f} \tag{22-6}$$

$$a>f$$

将它替换进式（22-5），得到图像点。

假设图像平面的位置不满足式（22-6），那么每条从P射出的光线经过透镜将会投影到图像平面上一个不同的点处。P将不聚焦，而是出现该点的一个圆形模糊表现。实际上，P点在理想距离d的前后有一个很窄的聚焦范围，换句话说，在以P为中心的一个小球体内的点都能在距离为d处的图像平面上得到清晰的图像。景深的效果是精确的，因为图像平面只聚焦那些在场景中很窄范围内的点，所有其他点都变得模糊了。

图22-6说明了该如何使折射光线不与透镜轴平行。考虑光线PQ。RC是与PQ平行但经过透镜中心的一条光线。RC与焦点平面相交于S，折射光线的构造如QS。注意，RC不是一条真实的光线，只是用来说明真实光线（用带有箭头的实线表示）形成的几何构造。在z=d处给定任何一个图像平面，容易计算这样一条光线与图像平面的相交点。我们能够轻松用这种方式证明从P点出发经过透镜的所有光线将会精确汇聚在相同的图像点上。

在分布式光线跟踪中，我们在图像平面的每个像素上采用抖动的采样模式。我们也为透镜导入一个棋盘分割来产生抖动采样模式。现在给定图像平面一个像素里的采样点和透镜上的采样点，我们需要确定进入场景中的主光线。在图22-7中，假设从图像平面到透镜的采样

光线是 $P'Q$、求与焦点平面的交点S和从S出发穿过透镜中心的光线SR。这给了我们所要的主光线的方向，这样主光线就是开始于Q点且与SR平行的光线。此外，很容易从上面的参数化光线方程中求出它。值得注意的是，在图中我们颠倒了箭头的方向。现在光线指向光线路径的相反方向，因为这些光线是跟踪进入场景中的光线。所以对于每个抖动的样本点（例如P'），可以求出透镜上的抖动采样点（例如Q），主光线QP的构造如上所示。这些主光线常在分布式光线跟踪中被跟踪，如上所述。最后像素值也是像素里所有抖动采样点的平均。

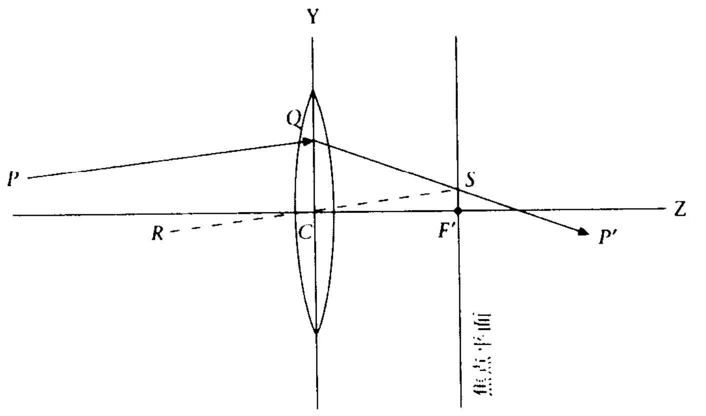


图22-6　折射光线不平行于透镜平面

476

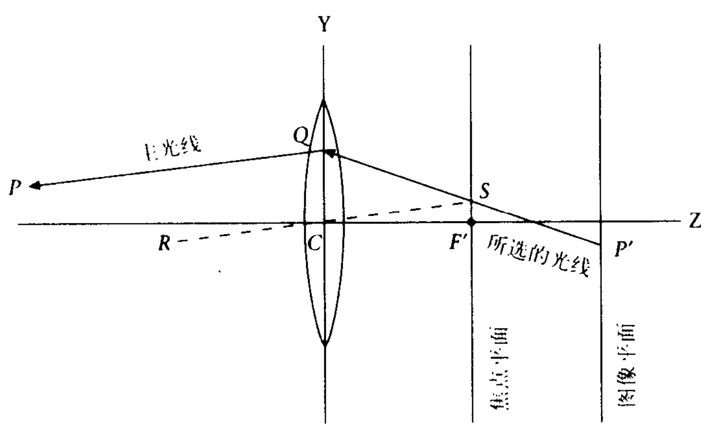


图22-7　从所选的光线中确定主光线

运动模糊

至此我们说明了分布式光线跟踪如何能迅速构造一幅图像。它是光亮度方程（对于经过透镜系统的主光线）的一个解，该光亮度方程允许景深、平滑和半透明的表面，并能解决走样问题，但是要求场景是瞬间静止的。它也可能扩展成时间上连续的解决方案，这样如果照相机或对象处在运动中也能渲染出运动模糊的效果。这是十分有用的。举例来说，在有些情况下，我们使用光线跟踪来产生动画序列中一组连续的图像。

不失一般性，让我们假设时间区间是[0，1]，任何运动都可以用参数$t \in [0, 1]$表示。现在

希望将每一条经过抖动采样模式的光线与一个时刻相关联。图22-8中给出了一个例子，这里像素被分割成16个子像素。时间间隔同样分割为16个区间，分别标记为1~16，这是随机分配给这些子像素的。现在假设t_{ij}是对应于第i行第j列子像素的时间片，那么对应于子像素的时刻是：

$$T = \frac{t_{ij} - 0.5}{16} \tag{22-7}$$

举例来说，$t_{12}=10$，$T=19/32$。现在我们添加一个随机抖动

477 （±(1/32)），这样T变成一个在范围$(18/32) < T < (20/32)$中随机选择的时刻。一旦T值是已知的，对象和照相机就移动到这个时间片上的正确位置，然后适当的主光线被跟踪进入场景，如上所述。

6	10	9	13
16	1	8	12
3	5	7	2
14	11	15	4

图22-8　在时间上采样

分布式光线跟踪总结

我们已经花了一些时间讨论分布式光线跟踪方法，因为它是光线跟踪本身外的在合成现实图像上的一个重要突破。它也允许我们去讨论一些额外的重要问题，例如走样和透镜效果。总的算法可以归纳如下（参见Glassner，1989，第5章）：

```
for each pixel {
  for each jittered sample point within each pixel {
    find the corresponding jittered time instant
    Move the objects and camera to their position at this time
    interval
    select the jittered sample point on the lens and find the
    primary ray
    trace the primary ray into the scene to intersect the first
    surface in the scene
    determine the reflection ray direction and a jittered sample ray
    around this, according to the BRDF of the surface
    determine the transmitted ray direction and a jittered sample
    ray around this, according to the BRDF of the surface
    determine a jittered ray direction for each light source
      and trace the shadow feeler rays
    call the ray tracing algorithm recursively
      from the reflected and transmitted directions
  }
  average the returned radiance values for each subpixel to determine
  the colour for the pixel
}
```

分布式光线跟踪如同我们所看到的那样，极大地改善了光线跟踪解决方案。然而，它本

478 身仍然不是光亮度方程的一个完整解。举例来说，它不能很好地处理漫反射现象。在下一小节中我们将研究另外的一个蒙特卡洛解——路径跟踪，它通过从每个像素中心发出多条光线确实提供了一组主光线方向的解，但是这里每条光线所派生出的反射光线和传导光线都沿着一条随机的路径，该随机路径基于相交表面的BRDF。（当然，通过在每个子像素上采用路径跟踪解，分布式光线跟踪和路径跟踪可以结合在一起。）

22.4　路径跟踪

Kajiya（1986）将多个计算机图形学光照模型统一到一个总的模型中，该模型基于光强度$I(x, y)$的渲染方程，光强度是从点y到x的，这里y和x约束在场景中的一个表面上（或者x可能在视图平面上，接收来自场景中表面的光线）。产生的积分式如式（22-8）所示，这里积分

是对表面S上的所有点，S包括封装整个场景的包围表面:

$$I(x,y)=g(x,y)\left[\varepsilon(x,y)+\int_S \rho(x,y,z)I(y,z)\,\mathrm{d}z\right] \tag{22-8}$$

$g(x、y)$是一个几何项，它要么为0（如果x和y彼此不可见），要么为$1/r^2$，这里r是它们之间的距离。$\varepsilon(x, y)$是从y发出到达x的光强度，$\rho(x, y, z)$是个无量纲的量，表示从z经由y到达x的无遮挡反射。因此对等式的陈述是：如果x对y是可见的，从y到x的光强度等于直接从y发出到x的光强度（如果y是一个光发射器）加上一个积分项，该积分项是在所有表面上对从任意其他点z到达y最后到达x的光线的积分。换句话说，$I(x, y)$包括从y发射出来到达x的光和总的从y反射到x的光。

如果x被看成是视图平面表面上的一个点，而且y被看成是表面与从COP出发的一条光线的相交点，那么式（22-8）的解对于每个这样的x会提供一个图像，说明环境中所有相关的相互反射光。

第3章的光亮度方程可以从式（22-8）中导出，而且我们将对该方程进行改造，变成式（22-9）的形式，方便后面的讨论。

$$L(p,\omega)=L_e(p,\omega)+\int_\Omega f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i\,\mathrm{d}\omega_i \tag{22-9}$$

光亮度方程（式（22-9））可以看成是对在某个特定位置、某个确定方向上光亮度的分解，分解成发射部分和反射部分。可以通过使用“积分算子”的思想更形式化地讨论它。积分算子应用到一个函数得到一个新的函数，新函数有算子形式所确定的性质。此时函数是$L(p、\omega)$，积分算子（记为R）将会返回反射成分L（排除了发射）。显然R可以被定义为:

$$RL(p,\omega)=\int_\Omega f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i\,\mathrm{d}\omega_i \tag{22-10}$$ 479

注意，将R应用到L的结果是产生在相同域中的另外一个函数，我们将它标记为$L^1(p, \omega)$。

使用这个记号，我们能完全以算子形式重写式（22-9）:

$$L=L_e+RL \tag{22-11}$$

这里域是通过参数（p、ω）给定的。算子理论允许我们重写式（22-11），对R的处理可以把它当作一个代数项:

$$\begin{aligned}L-RL&=L_e\\ \therefore(1-R)L&=L_e\end{aligned} \tag{22-12}$$

这里“1”是恒等算子，当应用到L时结果就是L本身。可以进一步将式（22-12）整理成:

$$L=(1-R)^{-1}L_e \tag{22-13}$$

最后算子表达式可以展开成幂级数:

$$\begin{aligned}L&=(1+R+R^2+R^3+\cdots)L_e\\ \therefore L&=L_e+RL_e+R^2L_e+R^3L_e+\cdots\end{aligned} \tag{22-14}$$

当然，R^iL_e意味着将R应用到L_e得到函数L^1_e，然后R再一次应用得到L^2_e。依次类推，直到应用i次。

现在来思考连续项的含义。首先考虑:

$$RL_e = \int_{\Omega} f(p,\omega_i,\omega) L_e(p,\omega_i) \cos\theta_i \, \mathrm{d}\omega_i \tag{22-15}$$

480 $L_e(p、w_i)$是对应在点p、方向为ω_i的直接光照（来自光源）的光亮度。因此RL_e是总的反射光的光亮度，从点p出发且方向为ω，这只取决于源于光源的光亮度。这可以由图22-9 说明，图中显示了两个光源的贡献，以及它们的效果是如何联合产生RL_e的。

我们可以写成$RL_e=L^1_e$。那么

$$R^2 L_e = RL^1_e = \int_{\Omega} f(p,\omega_i,\omega) L^1_e(p,\omega_i) \cos\theta_i \, \mathrm{d}\omega_i \tag{22-16}$$

L^1_e是从最初的光源出发只经过一次反射的光线，因此$R^2L_e=L^2_e$是相应于从最初的光源经“两次反射”后的光线的光亮度。这由图22-10所示，它显示了一条光线从光源出发经过两个不同对象的反射的情况。

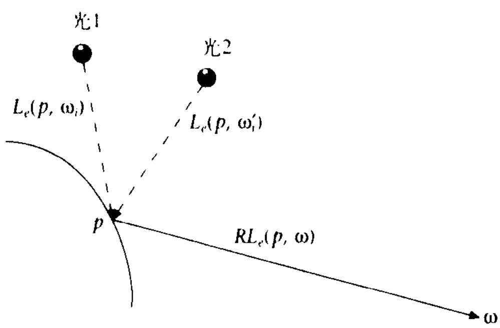


图22-9 图解RL_e

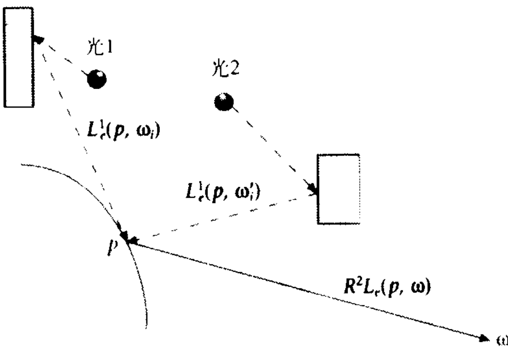


图22-10 图解R^2L_e

通常，$R^iL_e(p、\omega)$是点p所接收的方向为ω且经$(i+1)$长的路径从最初的光源到达该点的光对光亮度的贡献。$R^0L_e(p、\omega)=L_e(p、\omega)$不为零仅当表面本身是光发射器（因为它代表了经过1个路径回到光源）。

这样，我们就得到了一个结果明显的数学公式。光亮度$L(p、\omega)$可以分解为基于对象本身发射属性的光的光亮度、加上经过一次反射的来自光源的光的光亮度，再加上来自光源的经过两次反射的光的光亮度、并一直加下去。

现在考虑通过对连续项作有限项截取来得到光亮度方程的连续逼近。如果只将第一项作为解，那么我们将只渲染直接发出光线的对象。事实上第5章讲到的“平淡明暗处理”方法与此有些相似——每个对象被看成是一个光源，每个对象的反射系数为零。包括第二项所得到的近似值就像标准的图形管道：只考虑由光源本身所引起的直接反射，没有对象间的反射。

481 事实上标准的管道比这简单——因为光源只是点光源，典型情况下没有阴影。而且，一般点光源本身是不渲染的，只表现它们在其他对象上的效果。同样地，光线跟踪是另外的一个特别情况。把所有的表面都看成是完全漫反射器，Kajiya 还将光亮度方程简化为由Goral等（1984）引入的辐射度方程。

Kajiya 说明了该如何通过（马可夫链）蒙特卡洛积分来求解方程，并称其为“路径跟踪”。因为它包括对光子从表面到表面随机运动路径的跟踪——如在光线跟踪中那样，是一个“反向”的从视点到场景的路径。考虑光线从视点开始并穿过视图平面上的一个点（p），与第一个可见表面相交于点q。计算到每一个光源的随机光线（其分布取决于光源的发光特性），注

意光源不要求是点光源。对每条这种可见的光线我们计算在点q的光强度的局部贡献，并将它加到p点的光强度上。根据表面的材质属性（依据对象的漫反射和镜面反射的系数，以及透明程度）随机选择另外一条光线，确定它与环境（w）的相交点，重复这个过程，每次增加像素光强度。表面可能既有漫反射又有镜面反射的材质属性——这些决定了在表面相交点上将要发射的随机光线的分布。因此在每个光线的相交点上，光线总是被发射到光源处（以便得到局部贡献），然后再有一条光线以一个概率分布发射出去，该概率分布取决于表面的材质属性。不像标准的光线跟踪，在那里每个相交点上所有的光线（镜面反射和传导光线）都被跟随，这里只有一条这样的光线得到跟随。对于每个像素（或子像素）多个这种路径（N）被跟随，而且最后的光强度是所有这些的平均。Kajiya 对所有的例子使用 N=40。如果任何特别的路径其长度为$n+1$，则相当于截去展开级数第n项之后的结果。

这种蒙特卡洛路径跟踪的好处是明显的：各种表面类型都得到考虑，比较大的权值赋予了最重要的光线（主光线和到达光源的光线），而无需跟随那些对最后光强度帮助很小的路径。在标准的光线跟踪中，树的递归是相当深的，但在靠近树的叶节点处光线的贡献是很小的。

在这个方向上的研究主要集中在方差缩减技术上。每个像素的光强度由一个基于光线路径采样的估计量提供。**重要性采样**是一个强有力的方差缩减技术。这是一种在路径生成中考虑表面可见性和材质属性的方法。目前在这个方法上的研究已经有了相当大的进展（Arvo and Kirk，1990）。

处理漫反射的非常高效的方法也已经引入路径跟踪，这是通过在漫反射表面保持一个光照高速缓存来实现的。镜面相互反射占光亮度方程高频的绝大部分，它直接在路径跟踪期间采样。漫反射是变化较慢的部分，但确实是实际解中的基本成分。如果为充分表现相互反射而对表面上点的光照半球的大量光线进行采样，则其计算代价是无法接受的（当然每一条这种光线需要从它的相交点开始跟随）。 482

另一种做法是，因为表面的漫反射成分变化相对较慢，照明半球适应性地在少数几个点上采样，结果贮存起来（例如用一个八叉树），然后使用在插值中。这个模式分别由Warnock（1969）以及Ward和Heckbert（1992）引入，而且使用在光亮度软件（Ward-Larson and Shakespeare，1997）中。光亮度图像的一个例子如彩图P-1所示。

22.5 辐射度和光线跟踪的集成

光线跟踪如同我们所看到的，对镜面表面的仿真是非常理想的，而辐射度只适用于漫反射。真实场景包含这两种类型的反射表面，因此实际的解必须能模拟出这些。镜面反射效果可以通过后处理添加到辐射度中（Immel et al.，1986），但是如 Jensen和 Christensen （1995）以及Arvo和Kirk（1990）所指出的那样，这不能够正确地对漫反射-镜面相互反射建模，因为这种方法是单独执行每个过程的。

Heckbert（1990）为光子路径引进了一个标记方法来解释这一点。设L表示一个发光体，E代表眼睛，进入眼睛的光子开始于L且结束于E。一次与镜面表面的碰撞产生一个量S，与漫反射表面产生一个量D。那么我们用$L(S|D)*E$这种规则形式来表示所有眼睛可看见的路径，这里|代表“或”，*是一个乘法算符。举例来说 $LS\cdots SE=LS*E$。光线跟踪可以仿真 $LDS*E$ 或 $LS*E$这样形式的路径（见图6-1）。在光线跟踪中我们从眼睛开始“反向”追踪，从一个镜面表面到另一个镜面表面。然而，算法在经过一个漫反射表面之后就不能进行下去了，这是因

为从这种表面跟踪光线所产生的绝对计算复杂性。辐射度能模拟形式为$LD*E$的路径。现在如果我们在辐射度算法中增加一个光线跟踪过程，就能得到形式为$LD*S*E$的路径。

Heckbert说明了该如何为$L(S|D)*E$路径使用两遍双向光线跟踪方法得到完全解。第一遍是光跟踪。光子从光源散播出来（换句话说，光线经过光源分布到环境中），它们的能量如在光线跟踪中那样得到调整。光子遇到镜面表面的反射（或传导）按一般方式进行。当遇到一个
483 漫反射表面的时候，光子能量被存贮在一个与表面相关联的叫做自适应辐射度纹理（或rex）的纹理映射中。事实上形成这些初始光子路径的不仅仅是光源，而是包含全部漫反射（或部分漫反射）的反射表面都使用到了，正如在渐进细化辐射度方式中以亮度递减顺序。在这个阶段的结尾，rex将储存辐射度解。第二遍是“眼睛跟踪”——它由传统的从眼睛开始的路径跟踪来形成一个图像。然而，不同于光只服务于镜面表面的照明，所有那些有漫反射解的表面所关联的rex都得到使用——换句话说，在光跟踪阶段得到照射的表面有“阴影感知器”。光跟踪这一遍模拟具有形式为$L(S*D)*$的路径，因为许多镜面表面会在漫反射表面之前被跟随，从光源出发可能有许多这样的序列。眼睛路径能产生的序列形式是$DS*E$，因为光线或路径跟踪的使用和终止是通过使用来自漫反射表面中的rex的照明进行的，而不是通过光来进行的。因此我们把这些结合在一起就有了形式为$L(S*D)*S*E=L(S|D)*E$的路径。

双向光线跟踪下光线既从光源分布出来也从眼睛开始追踪，这是一种一般的技术，有许多不同的用法。在下一小节中我们将讨论一种或许是现今最流行的一般性全局光照：光子图。

22.6 光子跟踪

密度估计

在上面所讨论过的由Heckbert 提出的双向光线跟踪方法中，对光亮度的密度估计被储存在纹理中。一种显式的和完整的密度估计技术由 Shirley 等（1995）提出来，并得到 Walter 等（1997）的扩展。这是一个多阶段方法，它的优势是为实时漫游产生全局光照场景，其利用了统计密度估计技术。

方法包括对场景中所有的（多边形的）对象进行精细的三角部分。光子从光源开始跟踪，其能量的确定是根据发射器的能量分布。与对象的交是通过基于蒙特卡洛的跟踪进一步得到观
484 察，直到粒子最后被吸收。当一个对象表面上有一个撞击，信息就被存储在对应的三角形中。每个三角形可能有粒子撞击的一个关联分布，这个分布后面将用来估计光亮度密度函数。最后毗连的三角形若是根据一些误差准则足够相似的话就被合并在一起（这最后的步骤只是为了渲染的速度而减少三角形的总数），然后场景就能使用 Gouraud 明暗处理进行实时渲染了。

该方法产生给人印象非常深刻的具有照片真实感的图像——虽然带有许多的缺陷。在粒子跟踪阶段正确地使用了镜面反射和传导表面，但是由于最后的Gouraud 明暗处理，它们不是这样显示的。方法依赖于场景中的所有基本体素，这些基本体素表示为多边形。比较黑暗的区域不可能接受充足的粒子撞击来照明，而比较明亮的区域占据了所有的存储空间，因为在场景中发出的光的分布是非均匀的。

KD树

在前面曾提到过的照明高速缓冲存储器中，使用了八叉树作为空间数据结构以便储存漫

反射表面之间缓存的辐照度值。然而通常情况是，全局照明技术包括有限元方法，表面被分割成一个网格，并将照明信息储存在网格单元里面。如我们所看到的，这就是辐射度解的情况，这里必须使用一个大范围的不连续性网格化，以便得到最精确的阴影计算，它也用在我们上面所讨论过的双向光线跟踪方法中。Jensen（1996）引入了一种完全的全局光照方案，它一般存储场景表面上分布着的数十万光子，但是他的一个目标是将这些存储在与表面几何无关的一个数据结构中。这有两个好处：首先，解不依赖于网格化分辨率的无法预测的变化，避免了找到适当的网格化的困难。其次，网格化的方法其实只适用于有参数化表示的表面，举例来说，它不适用于不规则碎片形状，我们不可能提供一个这样表面的网格化（因为没有连续的表面）。Jensen 用KD树来实现这个目的（Bentley，1975，Samet，1990；de Berg et al.，1997）。我们在这一小节中给出对这些的简短描述，然后说明在 Jensen的 光子图方式中如何使用它们。

假设在d维空间中有很多点p_1，p_2，…，p_n，问题是对它们进行分类，并查询形状：求出位于与某个特殊轴对齐的平行六面体（或3D空间中立方体）中的所有点，或求出在某个点附近的所有点。在光子图的上下文中这些点将实际表现为撞击表面的光子。KD树实际上正是一个轴向对齐的 BSP 树。树的每个节点储存一个分割平面，由沿着其中一个坐标的中值定义。树的叶节点包含最初的数据点。

假如$p_i=(x_{1i}, x_{2i}, \cdots, x_{di})$。树的根将是分割超平面 $x=x_1$，这里x_j表示第j个坐标的中值。这会将最初的集合分割成两个相等的子集（习惯做法是如果一个值恰好位于分割超平面上，就将它加入到左集合中）。现在我们要递归地执行相同的过程到每一个左子集和右子集，除非每一个这些集合沿着x_2坐标被拆分。通常，坐标 $x_{k \bmod d}$用在第k个调用上。这个过程继续，直到树的每个叶节点包含一个数据点。下面是程序概要： 485

```
KDTree makeKDTree(int n, point_kd p[], int depth)
/*returns a kd-tree for the n k-dimensional points in p - assume all
indices start from 1*/
{
   if n==1 return a leaf containing p[1]; /*base case*/

   x = median of values of (depth mod d) coordinate in the points;
   pLeft is the set of points to the left of x and pRight is the
   set to the right;

   leftNode = makeKDTree(n/2,pLeft,depth+1);
   rightNode = makeKDTree(n/2,pRight,depth+1);
   /*assumes that the median splits the points exactly in two*/

   /*compose a new node and return*/
   return compose(leftNode,x,rightNode);
}
```

可以证明一棵 KD树需要储存量为 $O(n)$、其构造时间是 $O(n\log n)$。一个轴向对齐的平行六面体需要的时间是 $O(n^{1-\frac{1}{d}}+k)$，这里k是在输出中的点的数目。

光子图

光子是一束光流能量，也包含它所撞击的表面上的位置和它的入射方向。光子图是一个空间数据结构，特别是表示很多光子的KD树。在 Jensen（1996）所描述的方法中（也可参见

Jensen and Christensen，1995）创建了两种光子图，第一个具有非常高的密度，称为焦散光子图，第二个称为全局光子图。

焦散的光子图可以表示为一个路径，其形式为：$LS*D$；最后一束强烈的光波撞击一个漫反射表面。一个不错的例子参见彩图1-13。如果我们想像从某处看一个湖是一个漫反射表面（太远了，我们无法看见单个的水纹和波浪），那么撞击表面的光束将会形成表面上的一个图案。这种焦散效果公认是很难建模的（对于任何标准光线跟踪模式），因为在进入到眼睛里之前最后一种类型的表面是一个漫反射器。焦散是场景中高度复杂的反射类型，因此必须投入许多资源来捕获它们。Jensen的方法专门采用光子图来进行采样。光子从光源出发，只向镜面反射表面分布。当一个光子撞击到一个表面的时候，它是依照表面的 BRDF 反射出去的，并到达下一个镜面表面，直到过程结束于漫反射表面。因此对于焦散光子图，只沿着这种 $LS*D$ 路径。当与表面撞击的时候，光子被储存在 KD树中。

在Jensen and Christensen（1995）中，在光源上使用了光缓冲，这是为了分割镜面和漫反射表面。光子只会从光源送到镜面表面。

全局光子图的建立是通过从光源分配光子到所有的对象，可以以沿着任何类型的路径。它与焦散光子图相比有较低的密度，因为焦散光子图将在渲染过程中被直接可视化，而全局映射只是用于间接反射（即反射不是直接结束于眼睛）。全局光子图可以看成是光亮度方程粗略的全局解。有关阴影可能存在的地方也要储存，因为当光线运送光子到达某个表面的时候，它是持续在表面上的，因此标记为阴影光子。这个信息可以使用在稍后的渲染阶段中，来减少阴影感知器光线的数量，至少可用在间接光照的情况中。

486

在渲染阶段我们使用路径跟踪的一个变形。这当然是为了提供一组相关于视图方向集合的光亮度方程的特别解。考虑对式（22-9）右边积分，并重写为：

$$\int_{\Omega} f(p,\omega_i,\omega)L(p,\omega_i)\cos\theta_i \,\mathrm{d}\omega_i \tag{22-17}$$

分解入射光亮度如下：

$$L(p,\omega_i)=L_l(p,\omega_i)+L_c(p,\omega_i)+L_d(p,\omega_i) \tag{22-18}$$

这里L_l是来自光源的直接贡献，L_c是来自一镜面反射表面（表示焦散成分），L_d是来自至少经过一次漫反射的光子的贡献。

同样地，分解 BRDF 为：

$$f(p,\omega_i,\omega)=f_s(p,\omega_i,\omega)+f_d(p,\omega_i,\omega) \tag{22-19}$$

这里f_s是镜面部分，f_d是漫反射部分。

然后将式（22-18）和式（22-19）代入式（22-17）中（为表示上的方便不再讨论函数中的参数）：

$$\int(fL)=\int(fL_l)+\int(f_s(L_c+L_d))+\int(f_dL_c)+\int(f_dL_d) \tag{22-20}$$

这个方程使用在渲染阶段。它可以精确（如果方向是进入眼睛内）或近似（如果这是一条间接光线）地求解。我们连续地考虑每个项：

Direct $\int(fL_l)$。这是直接来自于光源的光亮度。对于近似解，我们使用一个来自全局光子图的光亮度的估计值。对于精确解，阴影感知器光线的使用仅当全局光子图中的信息无法明确给出这个点是否位于阴影中。如果受到阴影光子的包围，那么它在阴影中。否则，如果它

是一个混合区域，既有阴影光子又有非阴影光子，或者所有的都是非阴影光子，那么必须使用阴影感知器光线。

Specular $\int(f_s(L_c+L_d))$。这是来自镜面表面的光亮度，其解是通过路径跟踪获得的。 487

Caustics $\int(f_d L_c)$。这是来自漫反射表面上的焦散光亮度。这是直接从焦散光子图中获得的。

Diffuse $\int(f_d L_d)$。对于近似解，可以使用全局光子图中的信息。对于精确解，使用BRDF确定路径跟踪的一个方向集合。那些有可能对解有最大贡献的与这些一起都在路径跟踪中。

这样光亮度方程的解就是两项的和，其中一项是从表面点发出并射向眼睛的光的和（即式（22-9）中的L_e项），另一项是四个效果的总和：直接的、镜面的、焦散的、漫反射的，如上面所描述的。

很清楚这个方法需要有从光子图中从表面的点上抽取给定方向上的光亮度的能力。Jensen所用的一个估计如下所示：

$$L(p,\omega)\approx\sum_{i=1}^{N}f(p,\omega_i,\omega)\frac{\Delta\Phi(p,\omega_i)}{\pi r^2} \tag{22-21}$$

这里p为表面上的点，N是所发现的距离p最近的光子，入射方向为ω_i，且通量为$\Delta\Phi$（p, ω_i)。这构成一个半径为r、在表面上投影面积为πr^2的区域。因为光亮度是BRDF与辐照度的乘积，所有的这些乘积的和就是所需要的估计。

Jensen描述了系统一些典型的运行性质，多个场景的多边形总和达到大约5000个。使用过的最大光子图对于焦散性的大约包含390 000 个光子，对于全局映射大约需要166 000 个光子。在今天的计算机上这些场景可以在几分钟之内渲染完毕。由于光子图方法在现今渲染技术中的重要地位，我们上面概略地给予了介绍，有关于此还有许多内容应该写出来。最近的应用和发展放松了关于光只在真空中传导这个一般性假设，变成光在所涉及的媒体中传导（Jensen and Christensen，1998）。

22.7 小结

本章介绍了主要的基于光线的全局光照方法。我们说明了如何扩展光亮度方程，说明了渲染问题如何通过方程来解决，以及多种不同的渲染类型可以看作积分光亮度方程的级数展开的近似值。我们介绍了随机（蒙特卡洛）方法，特别是路径跟踪和光子跟踪。建议读者参考（Shirley，2000）来进一步了解这些主题。 488

第23章 虚拟环境的高级实时渲染

23.1 引言

让VE产生可信体验的主要需求之一是有高而稳定的帧频。有人可能认为这将会最终通过开发并投向市场的更快速和更强大的机器来达到。然而，模型大小和复杂度以及使用者的期待总是超过硬件所提供的能力。尽管硬件性能呈指数增长，仍然需要能降低所渲染的几何规模的算法，来保证不降低所得到的图像效果。通常采用三大类算法：可见性选择、细节层次（LOD）表示和基于图像的渲染（IBR）。基本上其策略有以下三个方面：

可见性选择。不经过渲染管道将多边形发送出去，除非它们有很高的可见可能性。

489 **LOD**。如果对象经过渲染管道送出去，那么要保证所送出的对象不要太复杂，在给定它们的尺寸和屏幕尺寸，以及对全局图像的重要程度等因素的情况下，只要能满足对它们的表示即可。

IBR。如果可以通过重复地渲染对象的一个图像（纹理映射）来代替对象的几何表示，那么就去渲染图像。

我们将简要地考察一下这些方法。我们也将会看到一种控制方法允许我们调整图像的逼真度（使用更有力的选择、比较低的LOD等），以便维持必须的帧频。

23.2 可见性处理

让我们从可见性处理开始。通常当模型变得很大的时候，从任意给定的视点处只有场景的一个子集是可见的。有三个方面原因解释为什么一些几何部分（比如说一个多边形）会是不可见的。它可能在视景体的外面、在观察者的背后或者是被更靠近眼睛的其他多边形所遮挡。我们在前面几章中已经看到了处理每一种情况的方法。在第10章中我们了解了该如何裁剪多边形和删除在视景体外面的任意部分。在第11章我们看到该如何判定一个多边形是面向还是背向视点的，也看到了可见表面的确定算法，通过它能准确告诉我们经过每个像素所能看得见的几何对象（第13章）。

然而，所有的上述方法都有一个共同的缺点。它们必须访问和处理在场景中的每个多边形至少一次。当模型变得很大、多边形的数目达到数百万的时候，这些技术就变得很不实用和十分浪费。在这一小节中我们将要看到的技术是将几何对象预处理到一个数据结构之中，该数据结构能保证在亚线性时间中查询，进而很快地拒绝那些对最后图像没有贡献的几何对象。

这些方法不是处理单个多边形而是成组地处理，举例来说，对所有的对象或对象集合进行处理。因为现在的计算机硬件中都有z缓冲或甚至完全的管道，这些方法的一般思想是求出有关的对象，让硬件来执行精细的可见性确定。

可见性算法基于输出可使用下列分类来描述（Cohen-Or et al.，2000）：

- 精确的可见性算法输出至少部分是可见的所有多边形的集合，且只有这些多边形得到输出。

- 近似的可见性算法输出一个包括绝大部分可见的多边形的集合，其中可能会包含一些隐藏的多边形在内。
- 保守的可见性算法输出一个包括至少所有可见的多边形加上一些非可见的多边形的集合。 490

显然，如果我们有一个精确集合，对于生成正确的图像来说，加在图形硬件上的工作负荷就会最小化。然而精确的选择方法通常开销很大，这就是为什么绝大多数可见性算法都设计为保守的原因。它们计算**潜在的可见对象集合**（PVS），如我们所说过的，包括所有可见的多边形加上一些非可见的多边形。这种过估计会稍稍增加一些处理，但是却能够保证生成正确的图像。一些算法允许得到一个近似解（Slater and Chrysanthou，1996；Zhang et al.，1997），当系统负载过高的时候这些近似解是有用的。它们可以用牺牲一些精确度来换取更好的实时性，这是通过忽略掉那些可能对最终图像没有太多贡献的对象来实现的。

视景体和背面消除

首要的一件事情就是确定场景的哪些部分不在视景体之内。

这项工作通常都是分层进行的（Clark，1976），它依赖于场景的存储方式。如果是在一个包围体层次结构中，比如包围盒（BB），那么我们进行如下步骤。从层次结构的根部开始，测试包围盒（BB）与视景体之间的关系，其结果可能是下面三种之一：

- 完全位于外部。在这种情况下，所有封闭在包围盒内的对象都在视景体的外面，所以对于这一帧它们全部都可以忽略掉，如图23-1中的节点A+B。
- 完全位于内部。在这种情况下，我们就没有必要再测试该节点下面的任何其他单元，如在图23-1中的D+E。事实上甚至不需要去做任何裁剪，我们只需要在无裁剪的状态下渲染所有包含的几何体。这通常是很快的，尽管今天的硬件不一定能确实达到很高的速度。 491

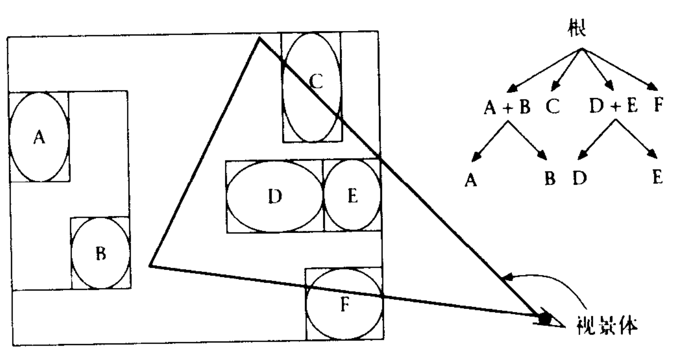


图23-1 对照包围盒层次结构的视景体选择

- 有一个相交点。在这种情况下我们分别测试根节点的每个子节点，如在图23-1中的根节点C和F。如果叶节点给出了一个相交点，那么我们需要对被包围的几何体进行有裁剪的渲染，如图中的节点C和F。

如果场景保存在一个层次空间细分中，而不是在一个包围体层次结构中，那么测试会稍微有些不同。举例来说，让我们以二叉树（如BSP树或KD树）为例。这里每个节点细分空间为两个，使用分割平面分离的不相交的凸子空间。对于视景体选择，同样我们从层次结构的最顶端开始遍历。一旦确定包围场景的整个子空间不完全位于外部或不完全位于内部，我们

就开始递归下降。在每个节点，我们相对于分割平面测试视景体。如果视景体完全位于一侧，那么我们能安全地忽视掉另一侧的子树。举例来说，在图23-2 中H_1和H_3节点的左孩子都不用遍历。如果平面与视景体相交，那么我们需要遍历两个子树，如在图23-2中节点H_4的情况。

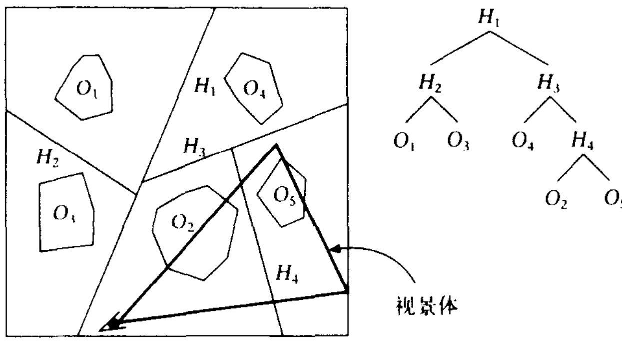


图23-2 对照空间细分的视景体选择

Slater和 Chrysanthou（1996）提出了另一种非常不同的方法。这是一种依赖于时序关联性的随机方法，这里对象与一个概率关联，这个概率说明它在可见集合中的可能性，该可见集合随着场景的改变或视点移动而得到更新。

视景体选择相对来说比较好实现，现在在绝大多数的渲染系统上都是存在的。

492 一般来说，对于任何给定时刻，在视景体中大约有一半的多边形是对最后图像没有帮助的，因为它们是背向观察者的。代替在第 11 章中使用的技术对它们的每一个进行测试，背面消除也可以在亚线性时间中用分层方法来做，如 Kumar 等（1996）所示范 的。在他们有关层次化背面计算的文章中，通过预先处理把模型根据法线和多边形物理的接近程度分割成群，整个大群集用层次结构来表示。空间相对于每个群被分割成三个清晰的区域。第一个区域定义为其中所有的面都是前向的，FrontRegion（所有多边形群的前半个空间的交集），第二个是所有群中的面都是背向的（BackRegion），第三个是剩余空间（MixedRegion）。在运行时算法相对于每个群的区域跟踪视点。对每个群，如果视点位于它的BackRegion（FrontRegion）中，所有的属于该群的多边形是背向的（前向的）。否则，它就是一个MixedRegion，我们需要检查子群。帧与帧的关联性可以用来进一步加速这个过程。

遮挡消除

这是至今最复杂和最有趣的消除类型，根据它的全局特性。它不能只通过考虑多边形得到确定，而是包括检查它们之间的相互关系。从任何给定视点，大量的对象受到靠近观察者的几何对象的遮挡。这里的任务是迅速识别并拒绝那些完全隐藏的对象。文献中有很多遮挡消除算法，由于篇幅关系我们无法给出这方面的一个完整综述，感兴趣的读者可以参考（Cohen-Or et al.，2000）。

最近又提出了许多消除方法试图求出潜在的可见集合，不是从单个点出发而是从单个区域出发（Cohen-Or et al.，1998；Durand et al.，2000；Schaufler et al.，2000）。之所以人们希望做到这一点有一些原因。因为结果对于在那个区域里面的任何视点是有效的，相同的 PVS 可以被重复使用，这样计算它的代价就由多个帧来分担了。其他的好处是解可以用来决定从磁盘载入什么几何对象（如果整个模型不能够放入内存中），或者在因特网应用中使用的

时候决定从远程服务器上下载哪些部分。

另一方面，针对点的算法比较容易编程而且运行比较快。它们也产生很少需要渲染的几何体，因为只计算对于特定视点的相关几何体。但是每个帧都要重复一次。

在这本书中，我们不准备进一步研究针对区域的可见性方法。我们将介绍一些针对点的方法。这些可以进一步被分类到对象空间(OS)方法和图像空间(IS)方法。无论对于OS方法还是IS方法，场景通常都是放在某种层次结构中，如包围体层次结构或空间细分层次结构中。通常我们需要执行一些能识别潜在遮光板的预处理过程。

图像空间遮挡

图像空间方法在图像的不连续分辨率中起作用，虽然也有例外。这里需要了解的是，最 493
后我们想要产生的东西是一个离散图像，因此求出在像素层次上的可见对象是充分的。

因为不连续特性，它们通常比较简单且具有较低的计算复杂度。它们能利用特殊的图形硬件来提高速度并有比对象空间方法更强的鲁棒性。同时，这些算法也能生成在误差控制范围内的近似解，这是通过把那些不重要像素能看得到的那些对象分类为被遮挡几何对象部分来实现的。这总是能提高运行速度。

很多小的和单个的不重要遮挡板的投影可以通过标准图形光栅化硬件聚集到图像上，去覆盖图像上那些很大一部分将被用于消除的内容。这些方法的另外一个优势是遮光板不必非是多面体，任何能够光栅化的对象都可以使用。

图像空间方法的概要大致是这样的：

```
traverse the scene hierarchy front-to-back (top-down) and at each
node N do
  compare N against the view volume
  if not outside
    test N for occlusion
    if not occluded,
      if N is a leaf node
        render the enclosed objects
        augmenting the occlusion structures
      else
        recurse down children of N
```

这个形式或许是理想的，因为它意味着当处理每个对象时要将它与那一点上所有聚集的遮挡进行比较。然而，对遮挡信息的连续更新是非常慢的，这就是为什么绝大多数算法要采用两遍处理。在第一遍中，它们渲染预先选择的遮光板集合，并创建遮挡信息，而在第二遍中，它们仅仅在遍历层次结构和保持遮挡信息不变的情况下进行消除和渲染。

对节点的遮挡测试的一个典型方式是将那些定义边界的面投影到图像平面上，并对它们每个进行测试。如果它们所有都被隐藏了，则节点是隐藏的。

最早提出来的图像空间方法之一是*层次化z缓冲区*（HZB）方法（Greene et al., 1993），
它可以看成是我们在第13章中已经看到的z缓冲区方法的一个扩充。开始的思想很接近上面的 494
伪代码形式。场景用一个八叉树存储，对它的遍历是从前向后（虽然其他的层次表示也是可能的）。因为可见的对象是扫描转换的，z棱锥被渐增地构造。每个节点在被进一步处理之前都要相对于z棱锥测试其遮挡性。

z棱锥作为一种遮挡表示，是一个分层的缓冲区，其每个层次有不同的分辨率。在最细的层次它就是z缓冲区中的内容，而比较粗的层次是将下面紧接着的比较细的一层在每个维度上

分辨率减半，并取对应的2 × 2窗口中最远的z值构成的。最粗的一层仅仅是一个值，对应于图像的所有z值中最大的。在对基本体素扫描转换的时候，如果z缓冲区的内容发生了改变，那么新的z值就会在棱锥中传播，一直到达最粗的一层。一个简单的例子如图23-3所示。

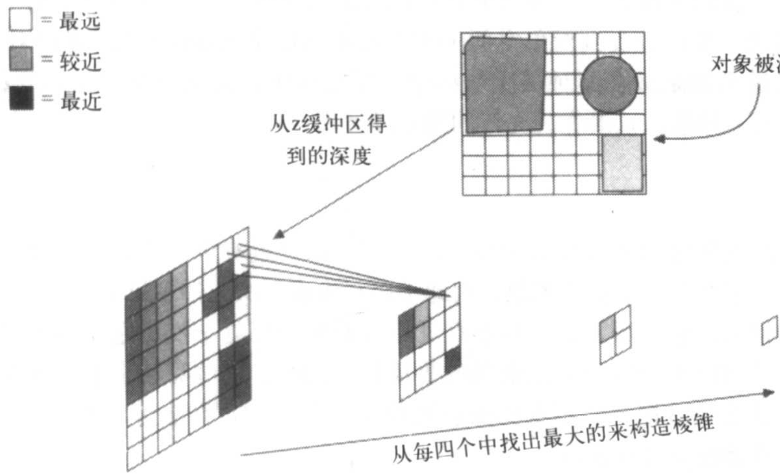


图23-3　层次化z缓冲区：z棱锥

为确定是否一个节点是可见的，将它的每一个面都投影到图像平面上，并相对于z棱锥分层次地进行测试。从棱锥最细粒度层次样本出发，该棱锥对应的图像区域覆盖屏幕空间中面的包围盒，将被投影面的最近的z值与z棱锥中的值进行比较。如果发现它更远，那么它就是被遮挡的，否则我们递归地下降到比较细的层次直到可见性可以判定为止。

为维护z棱锥，每当一个对象被渲染，引起了z缓冲区的改变，该变化就必须传播到比较粗的层次上。然而，这不是一种实用的方法，因为它需要对图形硬件进行连续的访问并处理更新。一个比较现实的实现是使用帧到帧的关联性并退回到两遍法（Greene et al.，1993）。在每个帧的开始我们首先渲染在先前一帧中可见的节点，在这一遍读取z缓冲区并构造z棱锥。最后使用z棱锥遍历场景层次结构来搜索遮挡，但是不更新它。

层次化遮挡图(HOM)(Zhang et al.，1997)是一个比较新的方法，在原理上与HZB相似。然而，这里的遮挡
495 问题被分解成了两个子问题。(1) 二维重叠测试，确定是否有潜在的受遮挡物在屏幕空间上的投影完全封闭于遮光板的聚集投影。层次化遮挡图就是用于这个目的。(2) 第二个是深度测试，确定是否有潜在的受遮挡物位于遮光板之后。

这也是一个两遍方法。在第一遍中建立遮挡图层次结构和深度信息，这是从一组“好的”遮光板中建立的，然后使用HOM遍历场景层次结构来决定每个节点的可见性。

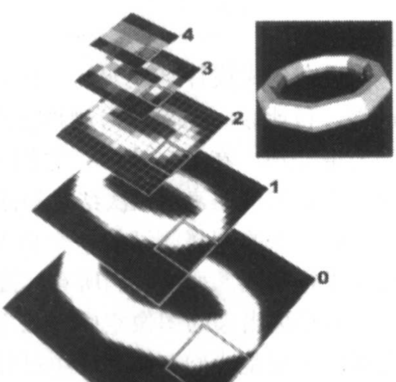


图23-4　遮挡图的层次结构（由UNC-Chapel Hill 大学计算机科学系的张汉松提供）

遮挡图层次结构的例子如图23-4所示。在最细的层

次上它仅仅是一个比特的位图，0（黑色）表示一个透明的像素，1（白色）代表不透明的像素，而比较高的层次上储存灰度值。

为了要建立HOM，我们需要从遮光板数据库中选择一组遮挡板并渲染到帧缓冲区之中。在这一点上，惟一需要的是占用信息，因此纹理生成、光照和z缓冲区都全部关掉。遮光板渲染成在黑色背景下的单纯白色。结果从缓冲区中读出来并形成遮挡图层次结构中的最高分辨率。比较粗的层次通过对2×2像素的正方形取平均来形成，这样就构成了一个在每个维度上对分辨率折半的遮挡图。纹理生成硬件能提供一些对求平均值的加速，如果图的尺寸大得足以保证硬件的安装成本。

当我们处理比较粗的层次时，像素不是简单的黑色或白色（遮挡的或可见的），而是可以用灰色色调的深浅来表示。在这样的层次上像素的强度给出了对应区域的不透明度。

对一个对象的遮挡测试首先是将它的包围盒投影到屏幕上并求出在层次结构中的层次，这里像素尺寸大约与投影方盒的区域一致。如果方盒与 HOM 的像素重叠，而且HOM是不透明的，这意味着方盒不能被消除。如果像素是不透明的，这意味着对象投影到了一个区域，在该区域中的图像被覆盖了。此时需要一个额外的深度测试来确定对象是否在遮光板之后。 496

在论文（Zhang et al.，1997）中提出了许多对象相对于遮光板的深度测试方法。一个最简单的方法是将一个平面放置在所有的遮光板之后。这个平面平行于最近的裁剪平面并且位于所有遮光板的最大 z 值的地方。当一个对象通过了不透明测试且最靠近的 z值远于这个平面的时候，我们就能判定它是被遮挡的。虽然这种方法快且简单，但它过于保守。另一种方法是被称为*深度估计缓冲区*的方法。在这种方法中，屏幕空间被分割成一组区域，对每个分割区域使用单独的平面。用这种方法，我们可以获得遮光板距离的一个更精细的度量，且伪可见的数目得到减少。

HOM方法相对于HZB方法的优势在于它也支持近似的可见性消除。有些对象可以被忽略，即那些经过遮光板小孔所能看到的很少一些像素上的对象。这可以通过设定一个*不透明度阈值*轻松完成，一个位于遮挡图中的像素在这个阈值上被认为是完全不透明的。这在图23-5中说明。图像中的矩形对应于一个对象包围盒的投影，该对象就是我们想要对其测试遮挡性的。如果要看图4中所覆盖的像素，我们会看到它们几乎是不透明的。它们不是完全不透明的，如同我们从图0所能看见的一样，该图中给出了一些小的孔。然而，如果它们的不透明度在阈值之上，我们能将对象分类为受到遮挡的，这样就不去渲染它，或对它做进一步的处理。

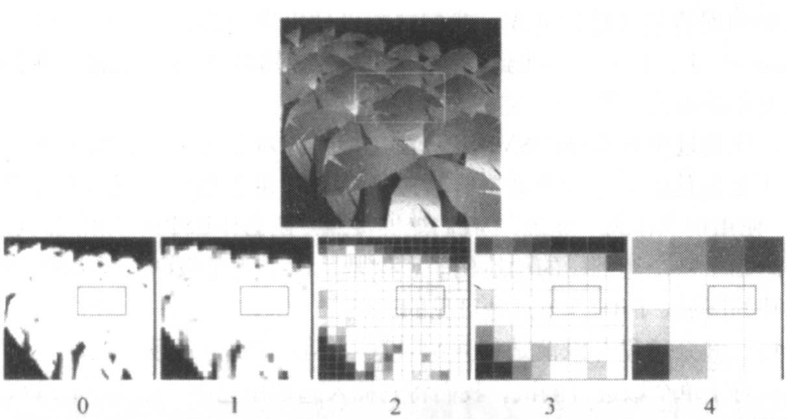


图23-5 层次化的遮挡图：近似消除（由UNC-Chapel Hill 大学计算机科学系的张汉松提供）

许多当前的图形卡提供基本的基于图像的消除，并由硬件测试。执行它的一个典型方式是通过在硬件中增加一种反馈机制，它能检查是否在扫描转换基本体素时z缓冲区会发生改变
497 (Scott et al., 1998)。如果场景存储在包围盒的一个层次结构中，我们能自顶向下地遍历层次结构，在每个步骤对包围盒的各侧执行这个测试。如果各侧全部都比已经在z缓冲区中的更远，我们可以停止对那棵子树的遍历，否则递归地继续下去。

对象空间遮挡

或许对于对象空间遮挡消除的最简单方法是使用单个遮光板的阴影墩。Hudson等（1997）介绍了一种基于这个思想的方法。不像我们在上面所看到的图像空间方法，它不聚集遮挡。对象可以仅仅被个别的遮光板遮挡，它使得该方法只对包含有大量凸多边形的场景有用。然而从正面效果来看，它不依赖任何特别的图形硬件，对于合适的模型能够快速地执行。还有两个其他的方法，分别是由Coorg和Teller提出来的（Coorg and Teller, 1996、1997），它们使用相似的概念，第二个方法也用一些遮光板融合。Coorg和Teller提供了一些非常有趣的值得研究的思想，但是在这里我们将专注于比较简单的阴影墩方法。

消除是分层完成的，因此场景需要存储在一个空间的层次结构中。为执行消除我们需要为每个帧准备一组好的遮光板。为使运行时对遮光板的选择更快，我们可以对模型进行预处理，并对每个空间区域存储一组可能是好的遮光板。这些可以用场景层次结构储存，或者用整个空间上的一个规则网格存储。然后在运行期间，我们基于视点选取一串遮光板并减少只存在于视景体中的那些。

从一个给定的视点，一旦有了遮光板，我们就能着手进行消除处理。对于每个遮光板构造一个阴影截头体。我们在第 14 章中给出过阴影截头体的定义，这里是相同的。阴影截头体定义了从遮光板后面的视点看去被遮挡的体积，完全落在其中的任何对象是看不见的，因而可以被忽略掉。我们自顶向下地遍历场景层次结构，且在每个节点N处都执行下面的步骤。相对于视景体测试N：如果它位于外面那么停止，否则轮流相对于每个阴影墩测试N。如果在任何点上发现它完全位于它们中一个的内部，那么我们就停止并对这一帧忽略N中存在的几何对象。如果N不与墩相交，我们继续向前并渲染所有在N中的几何对象。然而，如果N与墩部分重叠，我们需要递归并继续遍历N的孩子。

为了让这个方法有效工作，我们需要一个快速的节点到截头体的相交计算。假定空间的
498 层次结构使用轴向对齐模式已经完成，我们就能设计出快速的、专门的测试。详细内容可以在论文（Hudson et al., 1997）中找到。一个基本的（慢的）实现可以使用我们在本书前面部分讨论过的点对平面测试（第8章）完成。

退一步说，在看过第14章中的SVBSP算法之后，相对于每个阴影墩单独的层次结构节点的分类看起来不是最佳的。事实上非常容易应用SVBSP树思想到上面的遮挡方法上（Bittner et al., 1998）。使用视点作为"光源"我们能够按照"阴影体BSP树"相同的过程构造一个遮挡树。图23-6给出了一个遮挡树的构造例子，这是从三个遮光板O_1、O_2和O_3出发的构造。IN节点指示被遮挡的区域，而OUT节点是可见的。

给定遮挡树，其余过程与以前是相同的，惟一不同之处是对场景层次结构节点的遮挡分类方式。不是相对于单个墩进行测试，我们将它插入遮挡树之中。在 Bittner et al.（1998）中，这是通过单独地将每个节点的 6个多边形的边插入进树中并追踪它们到树的叶节点。我们在

第11章看到了如何将一个多边形插入进 BSP 树。我们说一个节点是完全可见的（被遮挡的），如果所有片段进入OUT（IN）区域。否则节点是部分遮挡的，我们需要测试每个孩子。

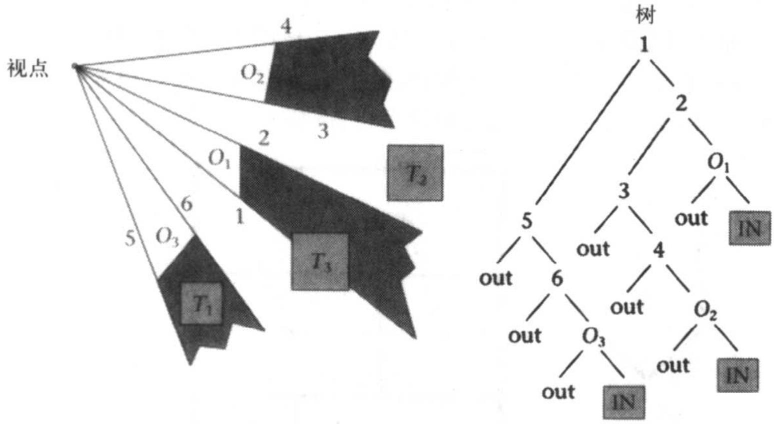


图23-6 使用遮挡树的层次化可见性

我们或许可以比单独地插入节点的每一侧做得更好。如果场景层次结构也是以二叉树的形式存在，就能使用 Naylor 等（1990）的BSP树合并算法，并通过执行在遮挡树和场景树之间的插入操作求出遮挡（Chrysanthou，2001）。

选择好的遮光板

求出正确的遮光板是任何遮挡方法中的一个重要部分，对于对象空间方法尤其是这样。由多边形相对的立体角是一个好的量度，但是它的精确计算是很慢的。一个经常采用的近似值如下所示（Coorg and Teller，1996）: 499

$$\frac{(-(a\cdot N\cdot V))}{|V|^2}$$

这里a是多边形的面积，N是多边形的法矢量，V是从视点到多边形中心的矢量（见图23-7）。

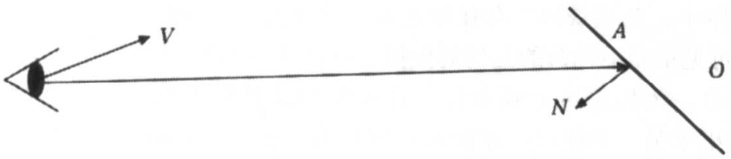


图23-7 遮光板选择

建筑场景

我们现在来看建筑场景这一个别情形，不但是因为许多通常的大模型是这种情况（建筑物、城市），而且因为它们密集并有良好定义的遮挡关系，这使得快速算法的构造来得比较容易。有两种类型的建筑模型：户内的和户外的。我们将给出分别对应这两种情况的算法。

单元和入口方法（Airey et al.，1990; Teller and Séquin，1991）是针对户内场景的经典的遮挡方法。它的工作机理不同于到目前为止我们所看到过的其他方法，因为它不追踪遮光

板来找出被遮挡的东西，而是通过入口（门和其他的开口处）来求出可见的东西。

在预处理过程中，模型首先使用BSP树分解成一些凸单元。主要的不透明表面（例如墙壁）用于定义分割，由此成为这些单元的边界。比较小的详细的场景元素被认为是“非遮挡的”，在这个步骤中被忽略掉。透明入口（例如门）被识别为单元的边界，并用它们来构成邻接图来连接细分的单元（参见图23-8中的例子）。粗黑直线是墙，用来分割单元，细灰直线是入口。在左侧，邻接图给出了哪些单元通过入口直接相连。

500

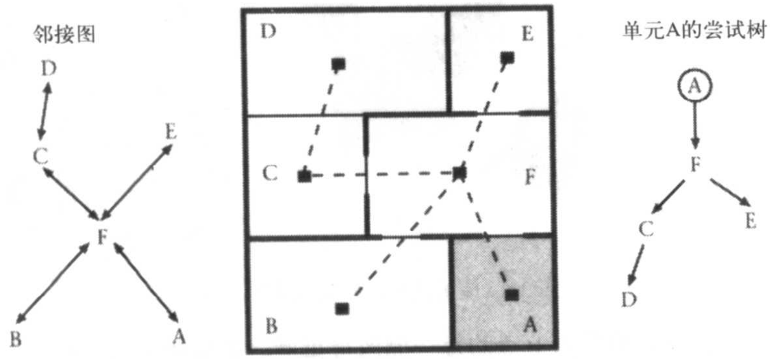


图23-8 单元和入口：邻接图和尝试树

*单元到单元*可见性的确定是通过测试是否存在连接一个单元中任一点和另外一个单元中任一点的视线。实际上很显然，如果存在一条从一个单元到另外一个单元的直线，它必须穿过一个入口，这样我们只需要确定是否入口在它们之间是可见的。对于每个单元，邻接图是利用来产生入口序列，该序列由视线“穿”在一起。举例来说，图23-8 右侧的树给出了一些单元，这些单元从单元A出发是可见的。对一单元内的任一给定点来说，那些从给定视点所在单元出发的视线所到达的单元中包含一个潜在的可见集合(PVS)。

在一个交互式漫游过程中，单元到单元的可见性可以使用观察者的视景体得到进一步动态消除，并产生可见场景数据的超集及*眼睛到单元*可见性。一个单元是可见的，如果所有这些为真：它位于视景体中，沿着尝试树的所有单元都位于视景体中，而且沿着尝试树的所有入口都位于视景体中，在视景体中存在穿过入口的视线，虽然我们可能决定只应用其中的部分测试。在每个可见单元中包含的几何对象接着被传送到图形管道进行渲染。

在上述方法中，为构造单元到单元信息所需要的预处理会是相当广泛的。Luebke 和Georges(1995)提出了另一种使用单元和入口的方法。惟一必须的预处理是单元和入口的生成并构造邻接图。其余的都在运行时完成。

从包含视点的单元出发，我们首先渲染在那个单元中的几何体（带有视景体选择），然后遍历到毗连的单元。为了进行遍历，入口的顶点被投影到图像空间。我们计算它们二维的轴向对齐的包围盒，叫做消除方盒，它们是入口的一个保守近似。任何对象经过入口，该入口投影位于消除方盒外面，则它是不可见的。当我们从单元到单元走过的时候，我们保持一个聚合消除方盒，它是连续方盒的交集。当这个交集为空的时候，我们停止沿着那个顺序的遍历，否则测试对象包围盒的投影以决定渲染哪些内容。图23-9给出了一个例子。在左图我们看见消除方盒以及它们的交集。在右图我们看见一个概貌，提醒我们这个方法等价于当穿越每个入口时对阴影体的收缩。镜子也能用相同的过程模拟。

有许多特殊方法专门处理户外的都市环境。对于户外的都市环境，其特殊之处在于我们可以把遮光板看成是2.5维的，即能使用它们的覆盖面积和高度来描述它们。而且，虽然我们没有一个容易定义的与入口相连的单元细分，但我们可以使用这个属性产生几乎一样快速的 501
方法。一个例子是 Wonka和 Schmalstieg（1999）的遮光板阴影。

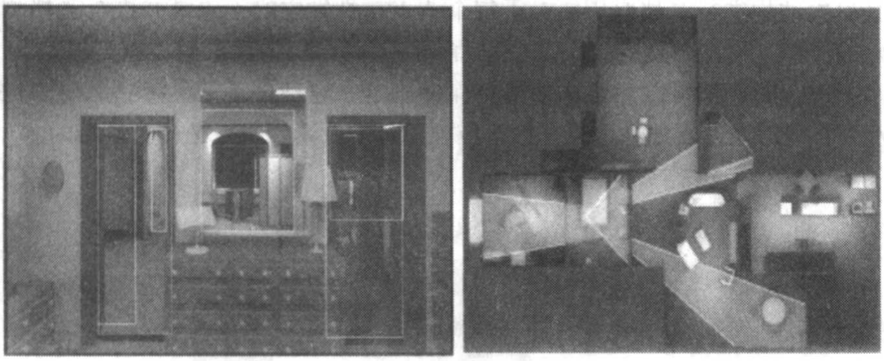


图23-9 图像空间单元和入口（由北卡罗来纳大学计算机科学系的David Luebke和Chris Georges提供）

将场景放入规则且与地面重合的二维网格之内。每个单元包含对应区域中的对象。在运行时遮挡信息如下构造。选择一组遮光板，使用硬件图形管道，计算它们的遮光板阴影并栅格化到一个深度缓冲区——消除图。遮光板阴影如图23-10 中显示。阴影平面是这样定义的：其顶点位于当前视点，并且穿过 遮光板顶边（v_a，v_b）。遮光板阴影位于遮挡板后面远离视点的阴影平面部分。对这些平面的栅格化，我们设置正交投影从模型的上方看下去，让图像平面平行于场景地面。我们还要定义视景体的范围和图像的分辨率，以便让每个像素精确地覆盖场景网格的一个单元。

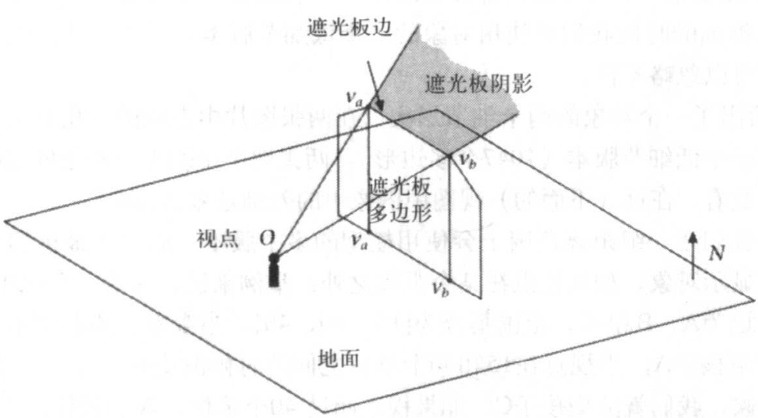


图23-10 都市模型中消除的遮光板阴影（由奥地利维也纳技术大学计算机图形学研究所的Peter Wonka提供）

当硬件对遮光板阴影进行栅格化时，z缓冲区中的值给了我们在每个像素上和在对应的单元上的遮挡高度。为了确定可见对象，我们穿过在视景体里面的二维网格单元，并测试相对于消除图中的z值每个对象的高度。如果对象有比较小的z值，那么我们可以忽略它。 502

上面所描述的算法不是保守的，因为一个部分受到遮光板阴影覆盖的像素将会设定为完

全被覆盖，如果它越过中点。为避免这些错误，作者提议将遮光板阴影的边缘以像素尺寸的某个比例向内收缩，同时在z方向上也施加一个校正。

Fallon 和 Chrysanthou（2001）提出了另外一个非常简单的实现都市选择的方法，这是遮挡图的一个简化版本。他们摒弃了耗费很高的图层次结构。对遮光板做渲染同时读取缓冲区，然后对场景层次结构进行遮挡遍历。对于每个节点，投影它的上边缘并进行遮挡测试是充分的。如果它被遮挡，那么节点的其余部分也将受到遮挡，因为我们假设遮光板是2.5维的（见图23-11）。上边缘的遮挡可以非常快地得到保守确定。它被位于最高y坐标处且与投影的x范围相同的水平直线所逼近。如果这条直线对应被覆盖的像素，节点是受到遮挡的。

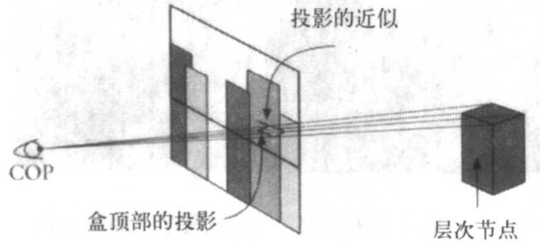


图23-11　都市模型的遮挡图

23.3 多分辨率表示

静态细节层次

细节层次（LOD）是一种简单的技术，它在对象远离观察者的时候降低对象的表示细节。其基本原理是如果所产生的图像只占屏幕的很少一些像素，则没有必要渲染太多多边形。这样每当视点足够远的时候我们就使用对象的一个低细节版本，所得到的渲染图像之间的差别是非常小的，可以忽略不计。 503

图23-12给出了一个对象的两个细节层次。在两张图片中左边的对象是右边对象（69 451个多边形）的一个低细节版本（1047个多边形）。两张图中中间那个对象的细节层次在另两个版本之间。注意看，在远（下面的）视图中图像中的差别是多么得小。

实际上，我们在一组距离范围上会使用模型的多个版本，而对于最远的一段范围可能就设为空，即不显示对象，如果视点在某个距离之外。举例来说，考虑一个LOD对象，它有3个模型，分别标记为A、B和 C，范围集合为[15，30，40]。当观察者的距离小于15个单位的时候，我们就渲染孩子A；当视点在15和30个单位之间的时候渲染孩子 B；当视点在 30 和 40 单位之间的时候，我们就渲染孩子C；如果视点超过 40个单位，我们就什么也不渲染。

对于简单LOD的一个问题是，从一个模型到另外一个模型的过渡通常是非常容易被觉察到的，因为它发生在单一帧上。在IRIS Performer工具包中提供的技术（Rohlf and Helman，1994）是在两细节层次之间做α透明性调配。取代突然的切换，我们把两个对象在一个过渡范围内同时绘制出来。两个视图使用透明性调配在一起，而透明性在多个帧上逐渐改变从而隐藏了转换。其缺点是在过渡过程中对象的两个版本都要绘制，这样这种过渡就只能是很少发生才有意义，否则就失去使用细节层次的优点了。

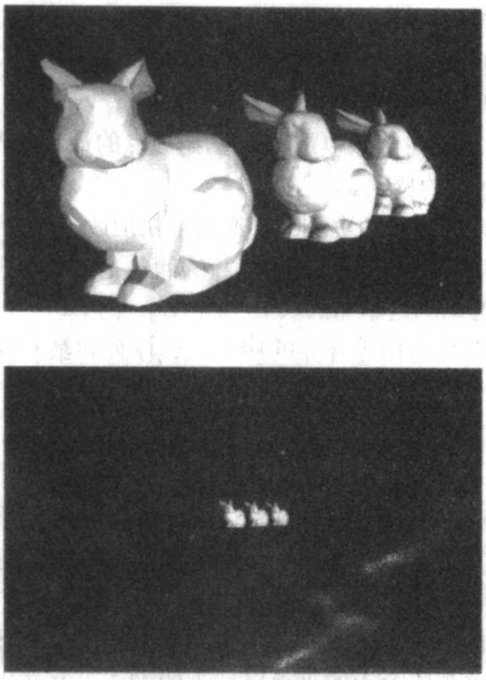


图23-12 细节层次示例（由UCL计算机科学系的Joao Oliveira 提供）

相对于在两个不相邻的几何集之间转变，另一种方法是在转换范围内将几何体从一个表示变形到另一个表示。下一小节中的渐进网格技术使用的就是变形的一个变种。 504

对细节层次转变的一个补充问题是，从最初的高细节几何体自动创建实际的细节层次。通常所使用的创建非常低细节表示的粗糙技术是用包围盒或者其他的简单包围对象替换每一个网格。在下一小节中我们介绍生成较低细节层次的网格粗化过程的一般性问题。

渐进网格

渐进网格背后的思想是非常简单的（Hoppe，1996）。不是在小而固定数目对象之间转变，我们从高细节网格到低细节网格创建连续形变，通过从网格中增加或删除边的方式并在观察者完全不能感知的情况下进行。该方法假设高细节对象是由三角网格组成的。通过重复执行操作对边做删除或添加，图23-13描述了这个过程。

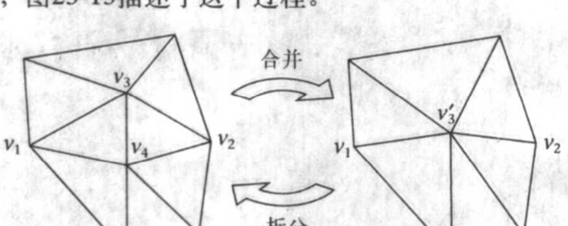


图23-13 网格细化的顶点拆分方法

当在运行系统中使用这个技术的时候，我们必须为网格确定多边形的目标数目。然后我们能应用一个或者更多合并或拆分，从当前网格移到目标网格。当对于静态细节层次转变的时候，我们通过在多个帧上制造变化来努力伪装这个变换。考虑图23-13中的例子。我们可以制作这

个合并过程，首先通过将顶点v_3和v_4向彼此移动，当它们进入某个误差范围之内时我们就用 v_3' 更换它们。当邻近的三角形扩展时两个三角形$v_1v_3v_4$和$v_3v_2v_4$逐渐变小，这看起来很平滑。

网格因而通过一个基网格M_0和一连串的拆分 $vsplit_0$，$vsplit_1$，$vsplit_2$，…，$vsplit_{n-1}$表示。每个*vsplit*标识要拆分的顶点（在例子中是v_3）、初始顶点的最后位置（在例子中是v_3'）以及被插入的顶点（在例子中是v_4）的位置。

细化可以被单独应用，但是实际上细节层次会很快地改变，所以 Hoppe 讨论了如何并行
505 地合并几个分离的动画。同时应用多个*vsplit*运算的问题是，一个顶点可能在附近的拆分中被移动两次，因此一个简单的动画是不够的。

对渐进网格技术的补充是选择边合并的序列，该序列将最初的网格变形成一个较低细节的网格。

网格的删减

我们所遇到的问题是选择哪些边来合并，以便让删减后的网格看起来与最初的网格相差不大。一个质朴的解决方案可能是简单地一个个删除最短的边。这的确给出了一个越来越简单的网格，但是结果看起来很粗糙。

问题是短边在高细节区域是很多的，这样它们就比在别处的大多边形有更重要的作用。一般来讲，网格删减应该首先删除大的平面区域上的细节，并在决定将要删除哪些边的时候考虑表面的局部曲率（Garland and Heckbert，1997）。

对这个主题的完整阐述超出了这本书的范围，但是图23-14 说明了问题的要点。最上面的一排给出了一个线框图和一个有69 451个三角形网格的平滑渲染。在下面三排的每一排中，左侧的一对是得自质朴的最短边删除算法的网格，而右侧的一对是得自考虑了局部曲率因素的网格。在第二排中，两个网格包含6 450个三角形。在第三排中的两个网格包含1 042个三角形。在第四排中，两个网格包含 540个三角形。注意在质朴算法中细节的损失大得多，尤其是在耳朵的周围。

506



图23-14 网格删减例子（最初的小兔模型由斯坦福大学计算机图形学实验室提供。网格删减由UCL计算机科学系的Joao Oliveira提供。对这些网格构造的算法可见（Oliveira and Buxton，2001））

帧频控制

细节层次和基于图像的技术（在下一小节中介绍）它们本身不能确保虚拟环境得到可靠的帧频率。首先，它们只是依赖于距离的，而不是依赖视图的，所以选择适当的范围在细节层次之间切换是困难的。然而，还有一个更严重的问题，因为范围只能被优化来匹配特别机器配置的帧频。如果有更快的机器，我们将会发现目标帧频被突破了，因此会希望扩大范围使之包括较多的细节[⊖]。

一般的问题是渲染时间是固定的，我们必须努力产生所能得到的最佳图像，可通过在场景图中元素间分配渲染时间来达到。Funkhouser 和Séquin(1993)将问题陈述如下。

定义一个对象元组（*O*、*L*、*R*）来表示场景对象*O*、渲染的细节层次*L*以及渲染算法*R*。可能使用的渲染算法有平淡明暗处理、Gouraud 明暗处理、Phong明暗处理、或各种不同类型的具有不同复杂水平的光照模型。对于每个元组我们可以定义*Cost*和*Benefit*启发式，这里*Cost*是所需的渲染时间，*Benefit*是衡量元组对场景总体感知的贡献的量度。于是问题变为：

最大化 $\sum_{S} Benefit(O,L,R)$

在下个帧（*S*）中所有可见的*OLR*三元组

在条件 $\sum_{S} Cost(O, L, R) < RenderingTime$ 下

渲染某个对象的代价依赖于很多因子，包括几何复杂度和所渲染的像素数目。这两者都是很难估计的（有时比渲染对象本身要难得多!），所以我们通常使用基于总的顶点数目的粗略估计。利益估计甚至是更复杂的。它依赖于对象规模、细节层次的精确度、用户的焦点、运动模糊以及滞后作用因素。我们将滞后因素考虑在内是要保证细节层次不会在两细节层次之间的边界上发生快速改变。利益的粗略估计只是投影屏幕的尺寸。

渐进网格方法对于这种情形是理想的，因为它提供了对 *L* 因子的微调功能——因为它能 507
够在完全网格表示和最低的一致的多边形表示之间改变。

最优化问题通过一个启发式算法实现——设*Value* =*Cost*/*Benefit*。Funkhouser 采用的思想是依照它们的值对*OLR*三元组进行排序，然后在链表的高端增加*L*属性或*R*属性，而在链表的低端减少，直到相同对象在相反方向上被改变了两次。这依赖于帧到帧的关联性，所以链表不需要进行重复排序。它不是优化问题的“正确的”解，它是NP完全的。

23.4 基于图像的渲染

在这一小节中我们将简要地讨论基于图像渲染的思想（IBR）。这个动机来自于这样的观察，即对表示复杂对象或场景纹理的渲染要比从场景图初始描述中读取场景本身并渲染它要快得多。换句话说，如果我们能渲染一个带有纹理的多边形，该多边形表示了一个复杂的对象，那么这将比直接渲染复杂的对象要快很多个数量级。这毕竟是纹理映射的一般动机。举例来说，我们可以渲染墙壁上的砖块的图像（纹理），而非逐个地对墙壁的每个砖块几何体进行渲染。

如何利用这一点来加速场景的渲染呢？当然，在一种极端情形中我们可以制作整个场景的某种纹理，然后仅仅渲染它。这就是“Quicktime VR”的基本方法，“Quicktime VR”是苹果电脑公司的开创性系统(Chen，1995)，该系统将场景的一组数字图像拼合在一起产生一个完全的3D环绕环境。用户能从多个预先设定的有利位置观察这个场景，系统通过图像插值从已

⊖ IRIS Performer有多个减缓这个问题的技术。最简单的是在帧频下降的情况下动态伸缩细节层次范围的能力。

经存在的图像集合合成新的图像。我们将会在下一小节中看到计算机图形学这种类型方法的另外一个例子，也可以参阅 McMillan and Bishop（1995），他们采用了一种场景的柱状投影。

这里让我们关注混合技术，即将标准的（以几何体为基础的）渲染管道与纹理使用结合在一起来节省渲染时间。这个思想通常称作*布告板*或*替代图像*。替代图像是对象图像的名字，该图像以纹理映射的形式使用。*动态的替代图像*通常是在运行期间通过对象渲染建立的，使用这个渲染来产生在透明的矩形多边形上的一个纹理。其思想是这样的，在运动视点序列中的连续帧之间有固有的关联性，同一个替代图像可以被重复使用在许多帧中，直到误差尺寸超过某个阈值。

Schaufler(1995)表述了这个思想（也可参看 Schaufler，1996）。比较早的时候，Maciel和Shirley(1995)曾经研究了一个类似的思想，但是在那里纹理是预先计算出来的，并与场景数据
508 库一并储存。Schaufler的方法是在运行时间计算 替代图像的，一经计算出来，同一个替代图像可以被重复使用在一系列帧中，只要替代图像中的图像和对实际几何体渲染所得到的图像之间的误差没有超出给定阈值。

对象的替代图像的创建过程如图23-15所示。设C是对象的中心，n是从C到当前COP的矢量。投影平面（P）是经过点C并以n为它的法线构造出来的。对象被投影到P，对象的包围区域也投影到P。包围对象的包围区域投影的最小矩形我们视之为替代图像的范围。这个图像于是作为一个纹理映射被创建在由该区域所给定的矩形上。多边形本身被标记为透明，所以它本身不会遮挡在它后面的对象，但是只有由投影形成的（不透明）纹理是这样的。其思想是这样的，对象只被渲染一次（形成替代图像），对于靠近当前 COP 的视点来说，可以渲染纹理来代替对象。

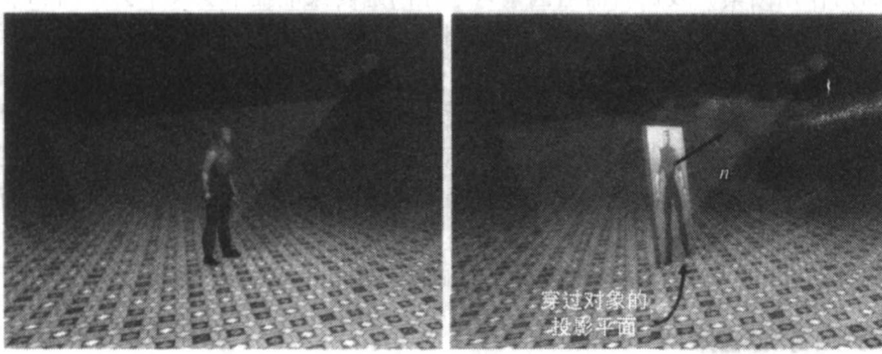


图23-15　创建替代图像（由UCL计算机科学系的Franco Tecchia提供）

纹理多边形可以通过使用被称为alpha通道的设施制作成透明，alpha通道在新的图形硬件上是很普遍的。我们知道像素由RGB值所组成。然而，典型地（举例来说，我们可以使用OpenGL得到）有一个被称为A值或alpha值。这是一个在 0.0 和1.0 之间的值，1.0意味着像素颜色是不透明的，因此对应的RGB值应该直接重写，无论现在在帧缓冲区中是什么。0.0 意味着像素颜色完全是透明的，这由意味着其“后面”的所有东西（例如已经在帧缓冲区中） 将会被看到（这样的话在alpha=0.0 时写 RGB 是没有效果的）。对于位于范围0.0~1.0的值，新的RGB 值是对在帧缓冲区中已存在的值的线性插值：$(1-\alpha)\mathrm{RGB}_{old}+\alpha\mathrm{RGB}$。因此当我们创造替代图像的时候，最初的矩形多边形应该设置成所有的alpha值为 0.0。然而，对于那些图像创建在多边形上的区域其alpha值被设置成1.0。注意，这个描述是对alpha缓冲区的典型使用，

而 OpenGL 允许许多其他的可能性。

纹理映射应该有什么样的分辨率？Schaufler 建议应该与屏幕分辨率成比例，该屏幕分辨率是由对象尺寸除以对象到 COP 的距离。因此，其他相同，比较远的对象需要比较低分辨率的纹理映射。

509

Schaufler 用于决定是渲染原对象还是对象的替代图像的原则是基于从视点看到的体素相对于像素的尺寸。如果我们设α_{texel}是体素所对着的角度，α_{pixel}是从相同视点像素对着的角度，那么替代图像在$\alpha_{texel}<\alpha_{pixel}$的情况下可以使用，因为这个条件表明在使用替代图像时的误差无论如何小于在屏幕分辨率上显示的误差。

当一个替代图像显示的时候，它所在的多边形（即纹理映射的多边形）总是经过了旋转，所以它的法线指向 COP。以这种方式图像总是面对观察者，纹理映射本身过程保证了应用适当的透视。然而，当然将会进入一个视点已经发生改变的片刻，视点相对于创建替代图像时的初始视点改变很多，这样替代图像就不再有效，需要重新建立。

当视点保持在相同的位置，只有观察的方向发生变化的时候，对象的投影在拓扑上是一样的，（也就是相同的顶点得到投影），而且可以使用相同的替代图像。然而，如果视点相对于替代图像创建时所对应的最初位置做了平移变换，那么就会引进一个很大的误差。这里有两种情况：第一是当视点沿着一个平行于对象中心的平面做平移的时候，第二是当视点移向或远离对象的时候。Schaufler 使用对象包围盒上点的投影对在这些情形时的最大误差给出了分析。在第一种情况中，如果我们从包围盒的一个角到其对角做一条对角线，想象视点沿着一条平行于此的路径滑动，那么从一个视点所做的投影重合的那些点当视点移动时将分离开。我们定义在这些点之间偏差的一个最大角度，如果它被超过就需要生成一个新的替代图像。同样地，如果视点在移动，比如向着对象移动，沿着垂直于包围盒一侧的矢量方向，那么对于包围盒的极值顶点存在相同的分析：在一个新的替代图像产生之前，最初位置和新位置之间的角度将会允许达到一个最大值。

上述讨论是依据对象的替代图像来阐述的。当然，思想是所有的复杂多边形对象应该有关联的替代图像，理想情况下绝大多数时间渲染器将渲染替代图像而非几何体。然而，这仍然有一个问题：如果有数万个这样的对象，比如形成场景的一个背景，在渲染这种大量个别纹理时仍然有内存和速度的开销。比较好的做法是以某种方式将对象变成群集，然后对每个对象群集形成总的替代图像。这一点由Shade等（1996）完成。他们注意到相对距离远的对象较之那些比较接近的对象需要较少的更新。因此可能聚集这种远的对象，并创建一个单一图像当作整个群集的替代图像。他们通过使用基于BSP 树的场景层次表示达成这一点。 510

首先有一个预处理阶段，在这个阶段中环境用一个 BSP 树来表示，其中叶节点是空间的一个凸区域，每个关联一组基本几何体素。最好是一棵平衡树。对这棵树有两次遍历。在第一次遍历中，每个节点被标记为三个状态之一。第一个状态是“消除”，如果它在视景体的外面。第二个状态是“几何”，如果节点的几何不能符合某个标准（例如位于距离视点的指定范围内，投影尺寸高于某个阈值）。第三个状态是“图像”，如果它既非消除也不是几何体。此时构成替代图像来渲染，而非几何体。这个图像替代图像可能已经存在了，如果在先前的帧中计算过它，但是我们需要确定它对于当前视点仍然是有效的。如果图像不存在或者它不再是有效的，那么我们通过对当前节点的子树渲染来计算它，要么当作几何体，要么从已存在的高速缓冲存储器读取。

第二次遍历是由前向后顺序，我们把每个节点当作一个几何体或者是图像渲染，究竟以何种方式渲染依赖于来自先前遍历的分类。

IBR 技术在最近的几年中引起了巨大的兴趣，并产生了许多研究成果（参见Popescu et al., 2000）。渲染图像是快速的，渲染复杂几何体则相对较慢。IBR的使用是一个极好的通过纹理生成硬件优化渲染的方法。举例来说，彩图23-16给出了一个带有数万虚拟人的都市人群场景。如果不使用 IBR 技术，在今天的硬件条件下对它进行实时渲染是不可能的。在某些方面，这个方法的意义在于标准的基于几何的渲染硬件是线性依赖于被渲染的多边形（三角形）的数目的。IBR 技术大大地减少这个线性依赖——在极端情形中，无论在最初的场景中多边形数量有多少，渲染一幅图像的时间总是相同的。因此如果整个场景和其间所有可能的视点基本上总是可以被渲染为一幅单一图像，那么这将彻底打破渲染时间对三角形数目的线性依赖。我们将在下一小节中转向这种极端解决方案。

23.5 光域

介绍

Levoy和Hanrahan(1996)以及Gortler等(1996)提出了一个基于图像的方法，该方法提供了第3章光亮度方程的一种强有力的解。光亮度方程是对光亮度$L(p, \omega)$的一个递归表达式，这
511 里 p 是任意表面点，而ω是所有方向的集合。任何一组特殊p和ω一起形成一条光线，因此我们可以把$L(p, \omega)$看成是初始点位于平面上的所有光线的集合。因此 L 域是一个五维光线空间。

然而，在“自由空间”中沿着一个给定的方向光亮度是常数，也就是说，由于与对象相交这里没有不连续性。因此光亮度在场景凸包外面的每个方向上是常数。图23-17给出了一个二维的简单例子。考虑起始于直线s结束于直线u的所有光线的集合。所有这样的光线与包含两个阴影方盒的场景相交在二维中，可以用（s，u）参数化表示。假设对于每个（s，u）我们可以（以某种方式）赋以一个光亮度值，也就是说，L（s，u）是已知的。现在要形成一个图像，我们将图像平面和投影中心放入场景，收集所有通过COP并与图像平面相交的光线。

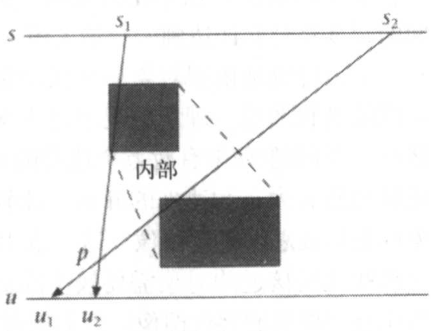


图23-17 自由空间中对所有光线空间的参数化

显然这种参数化只有对场景凸包外面的视图才有效，因为只有对于这些光线光亮度才是常数。在场景凸包内部任何点观察的视图都将是不可能的，因为光亮度沿着这种路径的改变是突然性的。

举例来说，假设我们需要针孔照相机从点 p拍摄带箭头直线所限定的视图“体积”内部的图像。那么，通过捕获位于箭头直线之间的所有光线的光亮度，我们需要构造这样的一幅图像。

为了构造光线（对于场景凸包）的完全集合，我们需要四个一样的由s和u定义的“光片”——一个如图中所示，另外一个有颠倒的方向，另外的两个是两条垂直直线分别位于场景的两侧。

不过，总的表示仍然是二维的，虽然有四个这样的二维表示。

这样的表示称为光域。光图有着相同的思想，尽管还带有一些附加的信息。

我们仍然停留在这个二维的例子中，对于s和u有单一的“光片”，光域光亮度会是s和u的一个连续函数$L(s, u)$。为了计算的目的，假设需要一个离散表示。两直线参数化由图23-18说明（二维的类比）。在s轴和u轴上构造网格，那么所有可能的直线一个顶点位于一个网格点处而另一个顶点位于另一个网格点处，这样的直线构成了所有这种直线集合的离散表示。

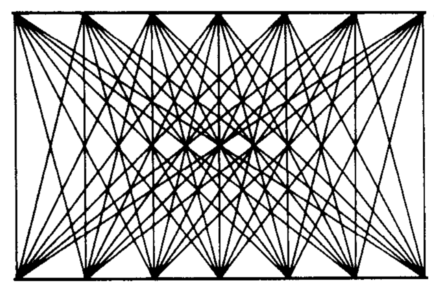


图23-18　光域的离散表示（一个光片）

对光域中的每一条光线的光亮度的估计是通过选 512
取s上的每个网格点s_i作为投影中心，同时将u作为图像投影平面（u的一个固定区间作为视平面窗口）。这样，与场景凸包相交的每条光线会有一个关联的光亮度，该光亮度取决于对一组通过透视投影所形成的图像的“渲染”。

对于3D场景，“两直线”参数化由两平面参数化（2PP）所代替，第一个由(s, t)参数表示，第二个由(u, v)参数表示。光域（事实上是它的一个子集）因而由$L(s, t, u, v)$表示，并被所有位于两平面之间的所有可能的直线离散化，这两个平面由其上的矩形网格所定义。st平面被分割成正方形，每个顶点形成一个投影中心，uv平面的矩形子集作为视平面窗口形成一个关联的图像。因此对于st平面的每个网格点有一个图像与之关联，对于每个（s，t，u，v）组合表示的直线，若与场景凸包相交则有一个光亮度与之关联。这描述了该如何形成一个光片。为了要覆盖所有的可能光线方向，我们需要六个拷贝——三个直角旋转，其在每种情况下有两个方向。

光域一旦构造出来，可以用来合成一幅来自虚拟照相机的图像，所谓的虚拟照相机不对应于在st平面上的照相机集合。一个新的图像可以通过采样直线的对应集合来构成，这些直线是经过视点并在所需要的方向上的直线。

光域方法是专为通过真实场景图像形成新视图的方法。假设我们获取了数字相片，在严
格的情况下，形成与st平面相关联的视点和方向的集合。这样的一组图像显然可以被用来构造 513
一个光域。一个虚拟的光域，也就是说用于虚拟场景的光域也能通过使用一些其他的渲染系统得以构造以便形成图像。这惟一可能存在的优势是渲染系统包括全局光照，而且还有漫反射表面和镜面反射表面的混合。这样光域方法大体上提供了在包括有光滑和镜面反射器的全局光照场景实时漫游的方法，这是用其他任何方法所不可能实现的。

在下个小节中将更详细地考虑一些问题。我们主要介绍由st和uv所参数化的光片。多重光片的扩展是明显的。

渲染：插值

假设st平面被$M \times M$网格所离散化，同时uv平面被$N \times N$网格所离散化。那么光域将包含M^2个透视图像，每个分辨率为N^2。这可以用图23-19中的二维实例说明。st上标记为COP的点将会有一个关联图像，其视景体位于两个比较粗的直线之间。所有汇聚于这个COP的光线因此将会有一个被这个图像决定的关联光亮度。图像可能是通过数码相机在真实场景中拍摄的，

或是给定COP和视景体对某个虚拟场景渲染得到的。COP依次在*st*平面上的每个点之间转变，这样完成完整的光片。整个过程必须对其他五个光片重复一遍，这些光片形成完整的光域，通过这种重复可获得一个完整的光线覆盖。

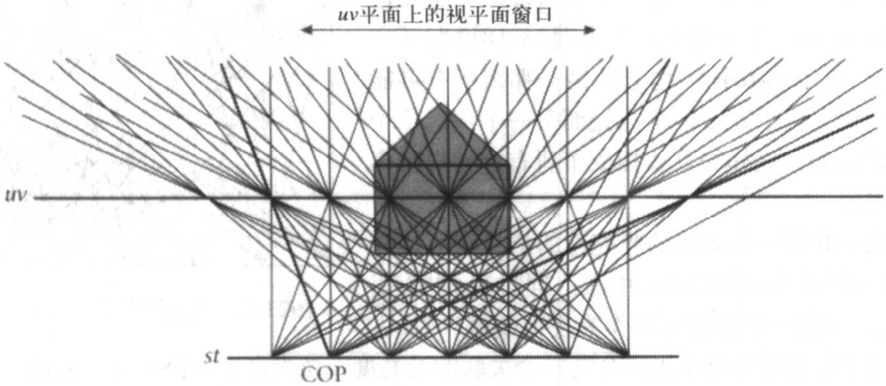


图23-19　对*st*平面上的每一个点创建一个图像

一旦光域已经建立，它就可能用来渲染从场景凸包外面某一点镜头位置和方向所得到的图像。首先我们描述它执行的基本方法。

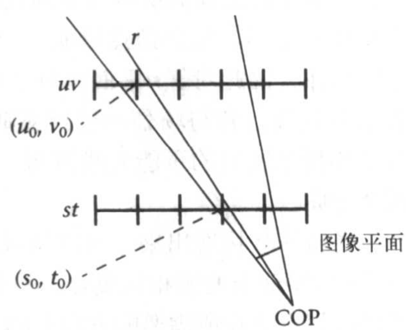


图23-20　为新视图合成图像

图23-20给出了*st*平面和*uv*平面以及新照相机的位置和方向。每条经过虚拟照相机的像素的光线将会与*st*平面和*uv*平面相交。举例来说，光线 *r* 与*st*平面相交于（s_0，t_0），与*uv*平面相交于（u_0，v_0）。因此相应于那条光线的像素会被设定为 L（s_0，t_0，u_0，v_0）。换句话说，每条从照相机产生的主光线被用来查找对应于光域的光线四维空间中最近的光线。

使用纹理映射这个方法可以非常有效率地实现。图
514 23-21给出了*st*平面上的一个网格点（s_i，t_j）和一个正方形，该正方形包含平面上的所有点，该平面由这个特殊网格点逼近。现在正方形投影在*uv*平面上的点 *a*、*b*、*c*和*d*。对应（s_i，t_j）有*uv*平面上的一个图像。这个图像可以当作一个纹理来渲染（s_i，t_j）的*st*正方形，其纹理坐标由*a*、*b*、*c*、*d*给出。因此，在此方法中，渲染对应于每个*st*网格点的正方形使用纹理映射对应于那个网格点的图像。这包括求出每个正方形的纹理坐标集合，但是因为邻近的正方形共享网格点，所需要的投影数目因而接近所绘制的正方形数目。当然，不是所有的M^2正方形需要渲染，这依赖于照相机的视景体。

实际上，这个过于简单的技术将会导致严重的走样。另一种可用的方案是使用四线性插值。在*st*平面的交点上有四个最近的邻近点，在*uv*平面上有四个最近的邻近点，如图23-22所示。首先考虑对*s*的插值，保持所有的其他参数固定。式（23-1）给出了一个关于*s*用s_0和s_1重心组合的恒等表达式。

$$s=\left(\frac{s_1-s}{s_1-s_0}\right)s_0+\left(\frac{s-s_0}{s_1-s_0}\right)s_1 \tag{23-1}$$

或

$s=\alpha_0 s_0+\alpha_1 s_1$ 515

且$\alpha_0+\alpha_1=1$

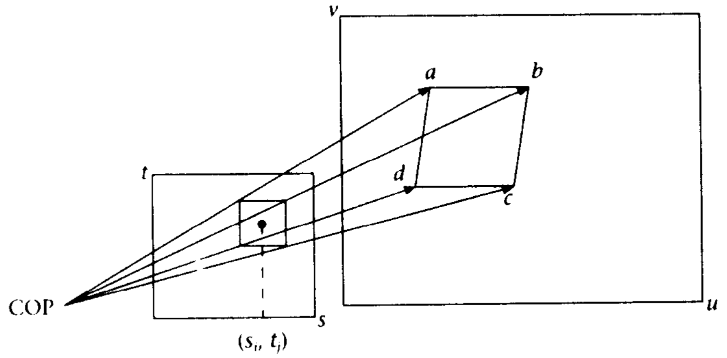


图23-21 使用纹理映射

四插值有这样的假设：L函数本身是仿射的（当然它实际不会是这样的），因此将它应用到式（23-1），我们得到：

$$\begin{aligned}L(s,t_0,u_0,v_0)&=\left(\frac{s_1-s}{s_1-s_0}\right)L(s_0,t_0,u_0,v_0)+\left(\frac{s-s_0}{s_1-s_0}\right)L(s_1,t_0,u_0,v_0)\\&=\alpha_0 L(s_0,t_0,u_0,v_0)+\alpha_1 L(s_1,t_0,u_0,v_0)\end{aligned} \tag{23-2}$$

对于每一个参数重复这个表达式我们有：

$$L(s,t,u,v)=\sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1}\sum_{l=0}^{1}\alpha_i\beta_j\gamma_k\delta_l L(s_i,t_j,u_k,v_l) \tag{23-3}$$

这里每个β_j、γ_k、δ_l的定义都与式（23-1）中关于α_i的定义类似，分别对应于t、u和v。利用类似于在渲染st正方形和uv正方形为多边形的过程中求插值纹理坐标的方法，我们可以很容易求得s、t、u、v的值。

Gortler 等（1996）指出这个方法不能够使用纹理映射硬件精确实现。对在(s_i, t_j)点处的四线性插值的支持在图23-23中延伸到 ABCD，因此附近的网格点将会有重叠支持。考虑包围网格点的六个三角形。Gortler 等说明如果这六个三角形的每一个都像前面那样用纹理映射来渲染再加上α调配（这里在网格点上$\alpha=1$，在其他三角形顶点上$\alpha=0$），那么这等同于在st上的线性插值和uv上的双线性插值。这提示我们结果从视觉上与完全的双线性插值是无区别的。 516

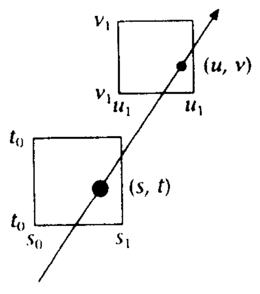


图23-22 四线性插值

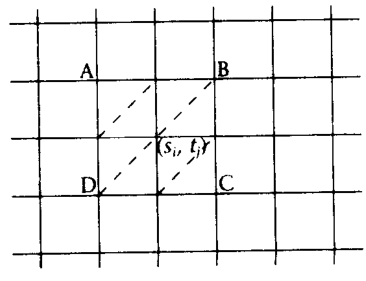


图23-23 四线性基的支持

Gortler 等（1996）指出除光域光线外保持一些几何信息能在光亮度估计中获得很大改善。图23-24 所示的是一条以COP 为原点相交于对象表面的观察光线。现在位于st平面上的最近的网格点如图中s_0，通常u_0和u_1会作为插值。很显然，我们可以从经过u'的光线s_0获得更精确的信息，因此好的插值将位于u_1和u_2之间。Gortler等提议存储接近场景几何的多边形网格以便处理这种深度校正。这个方法能得到与st分辨率相同的较清楚图像，因为对插值过程中的误差引入了一个补偿。进一步关于对渲染过程的加速、特别是质量和时间之间的平衡分析，可以参见 Sloan et al.（1997）。

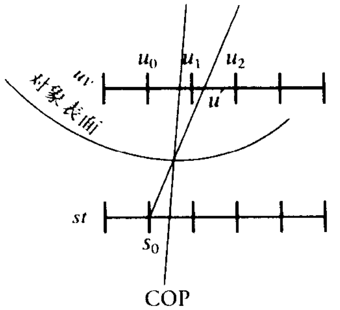


图23-24　深度校正

光域的表示

2PP有明显的计算优势、尤其是在重建中存储（矩形图像）和对渲染（尤其是纹理）硬件的使用。然而、这个表示不提供在四维空间中的统一的光线分布——换句话说，如果我们考虑在空间中的任意光线、不是所有光线在光域空间中以相同机率有相同程度的近似表示。实际上、其含义是所渲染图像的质量将是视点和观察方向的函数。靠近st平面中心的视点，方向与这个平面正交、将会得到比视点偏于一边斜方向看去时更好的图像——只是因为光线的数量和分布
517 在这个位置上将会是不同的。作为四维空间对2PP的完全分析可以在Gu et al.（1997）中看到。

Camahort 和 Fussel(1999)进行了三种可选光域表示的理论研究——如上所述的2PP、两球体表示(2SP)、以及方向和点表示(DPP)。Camahort等(1998)考虑了这些。2SP 包括在场景的周围放置一个包围球、并将球体分割成一致的网格。那么在球体上所有顶点之间的连接构成了参数化。DPP同样包括在场景的周围放置一个球体。随机地、均匀地选择球体表面上的一个点。这定义了一个自原点经过那个点的矢量。考虑垂直于那个矢量穿过球中心并被球体包围的平面（即在球体原点的圆盘）。现在在圆盘上选择一个一致分布的点集作为一组光线的原点，这些光线的方向与初始向量方向一致。所有这些光线的集合将在光线空间中构成光线的一致分布集合。每条光线由它的方向表示，它与圆盘相交于原点。一个不同的两球体参数化是由Ihm等(1997)引入的。该方法围绕场景放一个包围球体，然后在包围球体的表面上有一组小的球体表示方向。他们还为这个数据结构引入了一种小波压缩模式。

Camahort和Fussel(1999)提供了一个对这些不同模式的深度分析（对于一致性也有十分深入的讨论)，而且说明方向和点方法产生比较少的渲染偏移，校正也非常简单。

实际问题

有许多实际的问题需要考虑。首先，哪里是st和uv平面的理想位置？显然st在场景的外面，而uv应该经过场景的中心。理想情况是场景几何应该尽可能接近uv平面，这样网格点是（非常粗糙的！）表面几何的近似。实际上，这意味着光域不能够充分表示具有很大深度的场景。对近处表面表示所适当的光线分布并不很适合位于远处的表面。

对于M和N应该用什么样的分辨率？后者更重要，这是因为它确定了对场景几何近似的程度。uv平面上的分辨率愈高，这个表示的精度也愈大。因此实际上有$N>M$，且$N=256$和$M=32$已经被证明对于尺寸为256 × 256 的图像是足够充分的了。

假设我们使用这个分辨率，那么一块光片（slab）将需要2^{26}条光线。假如每条光线占3个字节，那么就需要192MB，因此对于整个光域（6光片），就需要超过 1GB的存储容量。我们知道这只将保证静态场景漫游和相对低分辨率的图像。显然需要一个压缩模式！Levoy和 518 Hanrahan（1996）使用了矢量量子化（Gersho and Gray，1992）。这是一个压缩模式，这里数据集分割成一些组，每个组由一个在训练阶段获得的矢量表示。每个矢量对应一个代码本，里面储存群集成员的指针。解码是一个非常快速的操作——在渲染的上下文中尤其有用。Levoy 和 Hanrahan 证明了使用这个模式的压缩比超过100∶1。

进一步的发展

最初的光域/光图论文发表于1996年。自那以后引发了极大的研究兴趣，我们这里只概述一些重要的进展。

如前面所提到的，光域模式的一个缺点是它最适合没有大深度的场景。Isaksen 等（2000）讨论了该如何克服这个问题，除了支持景深效果并说明光域如何可以用来作为自动立体显示系统的基础。后者的发展也已经在 Perlin 等（2000）的著作中有详细阐述。

光域方法显然是用于静态场景漫游而非用于场景中对象交互的。然而，Seitz和Kutulakos(1998)说明允许一定程度的交互，这是通过展现它是如何编辑一幅场景图像，传播改变到所有的其他图像并保持图像间总的一致性达到的。

表面光域是由表面发散的光线所组成的光域——原理上是对于表面上的任何一个点，离开那一点的具有关联的光亮度所有光线的集合。Wood等(2000)说明如何为真实场景构造、编辑和显示这种光域。一个重要的贡献是说明了这种压缩光域如何能以压缩的表示进行直接渲染。

如前面所述，光域表示理论在Camahort and Fussel(1999)中给予了讨论。Chai等(2000)提出了采样需求的一个完全分析。这包括一个最小的采样曲线，它提供了场景复杂度、输出分辨率和图像采样之间的关系。

总结

这一小节我们介绍了光域的思想，基于图像的方法提供了光亮度方程的解。LF方法几乎是此目的的一般“强力”——它用覆盖场景的所有可能光线的离散表示，然后通过图像渲染分配光亮度到那些光线。基于对图像的一个有限采样（用于颜色光线），它说明如何从 519 新的视点构造图像。有很多关于在插值模式的上下文中使用纹理映射硬件达到高效渲染的讨论。另一种表示光线统一集合的模式也得到了讨论，而且对其最近的进展给予了简短的讨论。

在这一小节中我们专注于 LF的基本思想，没有考虑所使用图像的来源。对光域的最广泛的使用是对真实场景的虚拟漫游。在文化遗产领域光域的一个极好应用在彩图P-3中给出。

23.6　全屏幕反走样

实时图形管道的一个基本问题是形状到屏幕的扫描转换，或是纹理的采样所造成的走样。举例来说，在“线段的光栅化”一节中，当画线段时因为连续的形状用正方形像素逼近时发生了走样。

克服这个问题的一个方法是全屏幕反走样。这包括在稍微不同的位置上多次渲染图像，然后构成最后的结果。这个过程也叫做**超采样**，因为它类似于渲染一幅更大的图像然后低采样来获得所要的图像。对于绝大多数渲染工具来说，如果屏幕渲染N次它的帧频就会下降N倍，但是该技术有时用硬件实现会有很小的或甚至没有时间上的代价。每一个个别帧是从照相机渲染的，这时照相机在源位置周围抖动少于一个像素的距离。当前的图形硬件支持合成过程，通过提供一个**集聚缓冲区**（Haeberli and Akeley，1990）达到。集聚缓冲区保持 RGBA（这里A是一个alpha通道）颜色值，如彩色缓冲区一样。我们不能直接渲染到集聚缓冲区，它必须经过像素区域拷贝得到访问。集聚缓冲区的使用可以用下面的步骤加以说明:

```
render screen from jittered position 0
load accumulation buffer with weighting 1.0/N
render screen from jitteredPosition 1
add to accumulation with weighting 1.0/N
render screen from jittered position 2
add to accumulation with weighting 1.0/N
...
render screen from jittered position n-1
add to accumulation with weighting 1.0/N
display completed accumulation buffer
```

在每个屏幕渲染之后它带有一个权值1.0/*N*，被增加到集聚缓冲区。当*N*幅图像被累积之后，这个集聚缓冲区就显示到屏幕上。彩图23-25给出了全屏幕反走样的一个例子。左边的茶
520 壶是一个单帧渲染，右边的茶壶是由八个抖动渲染组成的渲染。

在OpenGL中抖动照相机渲染可以用下列方式建立，它是从“红皮书”中摘录的一个例子（Woo et al.，1999）。

```
/* jitteredFrustum()
 * The first 6 arguments are identical to the glFrustum() call.
 * pixdx and pixdy are anti-alias jitter in pixels.
 */

void jitteredFrustum(GLdouble left, GLdouble right, GLdouble bottom,
   GLdouble top, GLdouble near, GLdouble far, GLdouble pixdx,
   GLdouble pixdy) {

   GLdouble xwsize, ywsize;
   GLdouble dx, dy;
   GLint viewport[4];

   glGetIntegerv (GL_VIEWPORT, viewport);

   xwsize = right - left;
   ywsize = top - bottom;

   dx = -(pixdx*xwsize/(GLdouble) viewport[2]);
   dy = -(pixdy*ywsize/(GLdouble) viewport[3]);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum (left + dx, right + dx, bottom + dy, top + dy, near,
    far);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
 }

 /* jitteredPerspective()
```

```
 *
 * The first 4 arguments are identical to the gluPerspective() call.
 * pixdx and pixdy are anti-alias jitter in pixels.
 */

void jitteredPerspective(GLdouble fovy, GLdouble aspect,
  GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy){

GLdouble fov2,left,right,bottom,top;

  fov2 = ((fovy*PI_)/180.0)/2.0;

  top = near/(cos(fov2)/sin(fov2));
  bottom = -top;

  right = top * aspect;
  left = -right;

  jitteredFrustum (left, right, bottom, top, near, far, pixdx, pixdy);
}
```

521

这些抖动照相机的定义与一般的glFrustrum函数和gluPerspective函数是非常相似的。下面的函数说明了这些照相机函数如何结合集聚缓冲区工作。在这个函数中displayObjects遍历程序数据结构或场景图来描述真实的场景。

```
#define ACSIZE8

void display(void)
{
   GLint viewport[4];
   int jitter;

   glGetIntegerv (GL_VIEWPORT, viewport);

   if (doAntialias) {
     glClear(GL_ACCUM_BUFFER_BIT);
     for (jitter = 0; jitter < ACSIZE; jitter++) {
       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       jitteredPerspective (50.0,
         (GLdouble) viewport[2]/(GLdouble) viewport[3],
         1.0, 15.0, j8[jitter].x, j8[jitter].y);
       displayObjects ();
       glAccum(GL_ACCUM, 1.0/ACSIZE);
     }
     glAccum (GL_RETURN, 1.0);
   }
   else {
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluPerspective(50.0,
       (GLdouble) viewport[2]/(GLdouble) viewport[3], 1.0, 15.0);
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();
     displayObjects ();
   }
   glFlush();
   glutSwapBuffers();
 }
```

23.7 VRML例子

静态细节层次和布告板 VRML97 都支持。图23-12 中的例子可用图23-26的VRML程序描述。Inline节点是一种在场景中包括其他文件而非在一个文件中描述所有内容的机制。文件bunny_high.wrl包含高细节对象，bunny_low.wrl包含低细节版本。我们也使用DEF/
522 USE机制与细节层次节点共享静态对象。这里有*LOD*节点的range域的一个单一值，这样在视点20.0 处我们从低细节到高细节切换，或反之。

```
#VRML V2.0 utf8

DirectionalLight
  intensity 0.8
{

Transform {
  translation 0.2 0 0
  children [
    DEF BUNNY1 Inline {
      Url "bunny_high.wrl"
    }
  ]
}

Transform {
  translation -0.2 0 0
  children [
    DEF BUNNY2 Inline {
      Url "bunny_low.wrl"
    }
  ]
}

Transform {
  translation 0 0 0
  children [
    LOD {
      range [20.0]
      level [
        USE BUNNY1,
        USE BUNNY2
      ]
    }
  ]
}
```

图23-26　VRML97 细节层次例子

VRML97也有布告板节点。轴旋转是用axisOfRotation域来设定的。如果轴设定为 0 0 0，那么对象就是面向视点的。也就是说，对象的局部坐标Z轴直接指向视点，而局部Y轴平行于照相机的 VUV。

图23-27中的例子给出了布告板的两个类型：轴旋转和面向视点。在图中的每一行中，左侧的方盒是一个面向视点布告板，中间的方盒不是布告板，而右侧的方盒是一个轴旋转布告
523 板。第一行的方盒给出了来自起始位置的视图。第二行对应于照相机偏转（关于Y轴的旋转），左侧和右侧的布告板看起来是相似的。第三行对应于照相机的前后俯仰（绕X轴旋转），面向视点布告板和轴旋转布告板之间的不同是很明显的。

23.8 小结

本章概括介绍了实时处理的一些当前技术。我们从可见性选择说起，所设计的过程试图使得最终送往显示器的多边形集合减到最少，只显示那些可能看得见的。一旦对象要被渲染，第二项技术就是试图减少它们的渲染时间，这是通过调整其细节层次来达到的。减少渲染时间的另外一个方法是使用替代图像和布告板。首先是纹理映射，使图像紧紧跟随所直接渲染的几何对象。显然渲染纹理映射带来极大的节省，只要纹理映射仍然是有效的（即在一定精度上很好地表示几何对象）。布告板也是纹理映射，它表示特殊复杂对象（例如树）当视点在交互过程中动态改变视点时，这些复杂对象面向视点的一面。如果能小心地做这些，VE的参与者用这种布告板不会注意到事实上几何对象是“平面”图像。然后考虑了另外一种完全不

同的渲染方式，即基于光域思想的渲染。其目的是从已存在图像的一个大集合来合成新视图。光域的一个主要优点在于它们的渲染时间是独立于最初场景几何的。最后我们考虑了一种简单的反走样技术，说明通过OpenGL访问的基础图形硬件的使用。

524

```
#VRML V2.0 utf8
Transform {
   translation 0 0 0
   children [
      DEF MY_BOX Shape {
         appearance Appearance {
            texture ImageTexture {
               url ["default.jpg"]
            }
         }
         geometry Box {
            size 2 2 2
         }
      }
   ]
}

Transform {
   translation 3 0 0
   children [
      Billboard {
         axisOfRotation 0 1 0
         children [
            USE MY_BOX
         ]
      }
   ]
}

Transform {
   translation -3 0 0
   children [
      Billboard {
         axisOfRotation 0 0 0
         children [
            USE MY_BOX
         ]
      }
   ]
}
```

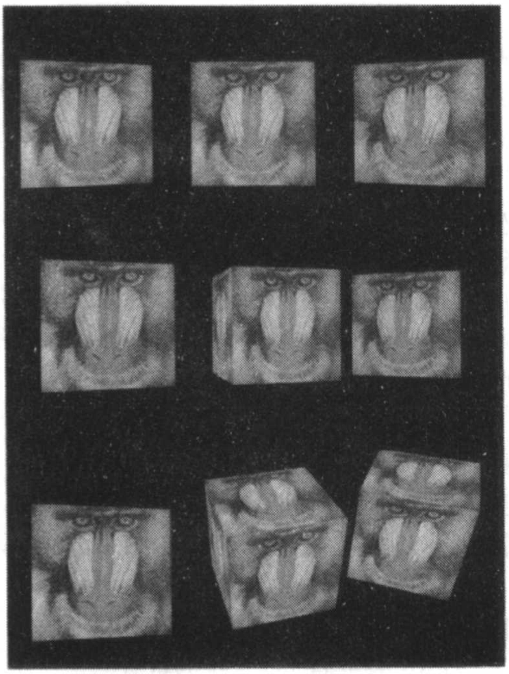


图23-27 VRML97 布告板例子

附录A VRML介绍

A.1 引言

虚拟现实建模语言(VRML，1997)是一个标准，它确定了描述3D 场景的文件格式，同时也是对这些场景进行动画描述、交互和仿真的一种机制。VRML场景一般是在VRML 浏览器中显示的，这样的浏览器例如 Blaxxun的Contact，它是在HTML浏览器中的一个插件(Blaxxun，HTTP)。VRML 被设计成分布在万维网上，VRML文件可以嵌入在HTML 文件中。VRML 文件用“.wrl” 这个文件名后缀识别，相应的MIME类型是*model/vrml*。

彩图A-1给出的是用 Blaxxun的Contact VRML 浏览器显示的Rhodes岛(COVEN，HTTP)上的 Lindos 寺庙的重建视图。屏幕底部的控制器允许使用者在场景中导航，场景里的一些对象是“活动的”，它们能响应用户在它们上面移动光标或者是用鼠标点击它们。因此，不像其他的3D文件格式（例如 VRML1.0）只描述静态场景，VRML97 能描述随时间演变的动态场景。场景中有一个蓝色的化身，位于图的中央，整个视图采用从化身肩膀上看过去所得到的效果。

在这个附录中，我们用三步对 VRML 给出一个概述。第一步简述场景图的基本内容，介绍场景图层次结构、几何描述和外观描述。第二步讨论动画，第三步讨论交互和脚本构造
525 能力。

除此之外，你将会发现在其他章中所涉及的一些节点的额外讨论。尤其是：

- 在“ VRML97中的光照”中的材质；
- 在“VRML97 例子”中的照相机；
- 在“使用VRML97”中的几何与变换；
- 在“VRML97 例子”中的纹理生成；
- 在“VRML 例子”中的交互和媒体；
- 在“VRML 例子”中的 LOD 和布告板。

A.2 基于VRML 的场景描述

节点和域

一个VRML 文件包含一组描述场景的节点。每个节点是由一些域组成的。例如，VRML有一个圆锥体节点，允许创作者用四个域描述一个圆锥体，这四个域分别是bottomRadius、height、side和bottom。圆锥体的域的描述如下面所示：

```
Cone {
  field SFFloat bottomRadius 1
  field SFFloat height       2
  field SFBool  side         TRUE
  field SFBool  bottom       TRUE
}
```

对于每个域有一个域的类型（此处为SFFloat或SFBool——浮点数或布尔值）、域名字以及域的默认值。在指定圆锥体时，bottomRadius和height的作用应该是显然的。域bottom和side定义圆锥体几何体是否有底部或侧面。举例来说，设定底部为FALSE，则圆锥体就是底部开放的。 526

当我们真的将一个圆锥体节点写入VRML文件中的时候，只给出域名和值，如下面例子：

```
Cone {
  height 5
  bottom FALSE
}
```

这里，因为没有bottomRdaius及side被设定，我们就假定它们使用缺省值（分别是1 和TRUE）。

域可以分为四种类型：一般field、eventIn、eventOut和exposedField。我们将在下一节中对每个域的使用给予介绍。暂时我们可以认为仅用fields和exposedFields就足够描述静态几何对象的了。域也有单值域（用SF前缀表示）和多值域（用MF前缀表示）之分。进一步增加复杂度，域的类型可以是单节点或多重节点的（分别用SFNode或MFNode表示）。用下面的例子将很容易说清楚。

技术细节

每个VRML97文件以“# VRML v2.0 utf8”开始。下面的代码段是关于绿色材质的球体的描述：

```
#VRML V2.0 utf8

 Transform {
   children [
   Shape {
     appearance Appearance {
       material Material {
         diffuseColor 0.1 0.7 0.2
       }
     }
     geometry Sphere {
       radius 2
     }
   }
  ]
}
```

符号“#”除对于第一行外任意行表示一条注释，将会被VRML浏览器忽略。

VRML97被设计用在互联网上，因此它总是集成在网页中。这通常是使用插件的方式，下面的一段HTML代码说明VRML是如何嵌入到网页中的。需要注意的是，渲染窗口的尺寸以及由此产生的照相机视图的纵横比是在HTML中而不是由VRML定义的。 527

```
<html>
  <head>
    <title>VRML - Example1</title>
  </head>

 <body>
   <h1>VRML - Example1 </h1>
     <center>
```

```
        <embed src="example1.wrl" border=0 height="300" width="400">
      </center>
    </body>
</html>
```

形状和几何

3D场景的基本组件是一组形状，由它定义可见对象。每个shape有一个几何域和一个外观域。几何域包含一个Geometry节点，外观域包含一个Appearance节点。Shape节点的定义如下所示（注意域类型）：

```
Shape {
  exposedField SFNode appearance NULL
  exposedField SFNode geometry NULL
}
```

在先前的例子中，几何节点是一个球体：

```
geometry Sphere {
  radius 2
}
```

外观节点包括一个带有漫反射颜色的材质：

```
appearance Appearance {
  material Material {
    diffuseColor 0.1 0.7 0.2
  }
}
```

基本的几何节点包括Box、Sphere、Cylinder、Cone和Text。图A-2给出一些基本例子，说明在每一个对应的节点中的主要域。

最灵活的几何节点IndexedFaceSet我们在“使用 VRML97”一节中描述过。

外观节点定义了对象的外表，它由三个域组成：一个Material节点、一个Texture节点和一个TextureTransform节点。

Material节点的细节在“VRML97中的光照”中介绍。Texture和TextureTransform在“VRML97 例子”中介绍。

组、变换和场景图

一旦一组对象用几何和外观创建，它们必须被放置在场景里。Transform节点就是为这
528 个目而设的。变换的细节内容在“使用 VRML97”中介绍。

构造场景图的一个有用工具是Group节点。这在场景图中提供了一个节点，它没有什么固有特性。其作用与变换节点相同，都有一个单位变换。组是一个能用来识别具有特定任务的一个对象集合，它尤其在场景图中元素需要共享时非常有效。

VRML97 支持节点共享以便降低文件规模和最后在内存中场景图的规模。场景图中任何节点都可以通过使用DEF这个关键词指定为共享节点。一个共享节点由此可以在今后通过使用关键词USE来重复使用（另外一个实例和更详细的关于如何使用DEF和USE的介绍可见“使用VRML97”）。DEF和USE两个关键词都可以嵌套，这就允许我们重复使用某些元素来构成非常庞大的模型，而重复元素可以以某种紧凑的方式存储。在图A-3所示的例子中DEF ONE_BOX语句创建了一个简单的具有纹理的方盒，并标记场景图中这一部分以便将来复用（见“VRML97 例子”关于VRML97 的纹理映射描述）。当使用ONE_BOX的时候我们得到场景

图当前点处的节点的拷贝。注意我们是如何在一个Group节点中首先将这些打包，并赋予这个对一个名字 PAIR_BOX。然后我们重复使用PAIR_BOX来创建所有的四个方盒。 529

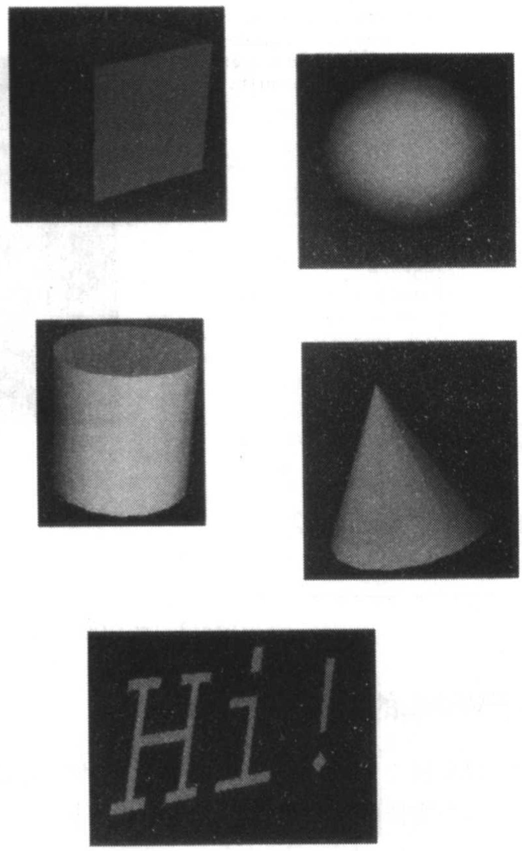
```
Box {
   size 2.0 2.0 2.0
}

Sphere {
   radius 1.5
}

Cylinder {
   height 2.0
   radius 1.0
}

Cone {
   radius 1.3
   height 1.8
}

Text {
   string ["Hi!"]
   fontStyle FontStyle {
      family "TYPEWRITER"
   style "ITALIC"
   }
}
```



图A-2 五个基本几何节点的例子

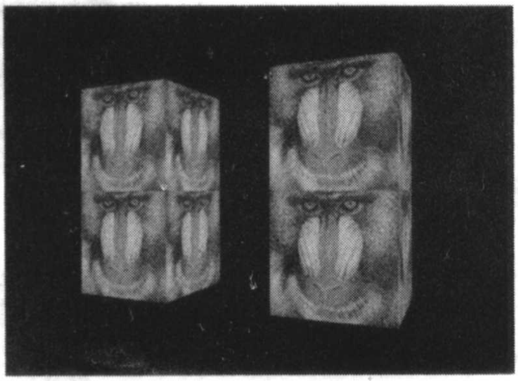
注意，虽然任何节点都可以用DEF来定义，但是必须注意节点在哪里使用。VRML的节点都是隐式类型。如果一个域是一般类型SFNode或MFNode，使用的情形暗示一个子类型的存在。这样只有Material节点可以被Appearance节点的材质域作为值来使用，使用一个由对应的DEF附着在Geometry节点上的节点将是违法的。

光

光节点在场景中提供照明。光与对象的材质交互从而产生在 VRML 浏览器里所看到的最后图像。它支持三种类型的光：DirectionalLight、PointLight和SportLight。DirectionalLight定义平行照射的光线，比如日光。PointLight定义从指定点出发向所有方向照射的光。SpotLight定义从某个点出发射向某些特定方向的光，同时可以指定在空间中的光强度。SpotLight和PointLight的效果随距离而削弱。彩图A-4给出了每种光 530
的一个例子。

```
DEF PAIR_BOX Group {
 children [
  Transform {
   translation -2 0 0
    children [
    DEF ONE_BOX Shape {
     appearance Appearance {
      texture ImageTexture {
       url ["default.jpg"]
      }
     }
     geometry Box {
      size 2 2 2
     }
    }
   ]
  }
  Transform {
   translation 2 0 0
   children [
    USE ONE_BOX
   ]
  }
 ]
}

Transform {
 translation 0 2 0
  children [
   USE PAIR_BOX
 ]
}
```



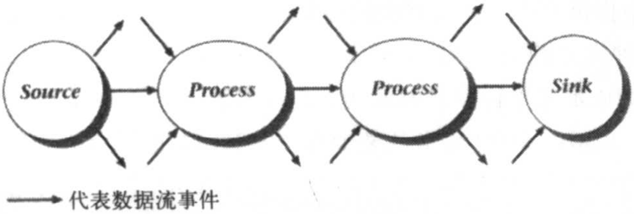
图A-3 DEF/USE和Group的例子

A.3 基于VRML的动画描述

VRML97在网上成功的最重要因素之一是它定义动画模型的能力。简单的动画可以用节点定义，并不需要任何脚本。数据流模型用来描述动画随时间的演变。

抽象数据流

数据流的抽象视图如图A-5所示。图结构是由节点和弧构成的，这里节点表示数据处理，弧表示数据从一个节点到另一个节点的转移。某些节点以稳定的速度不断地产生值。在一般的模型中它们可能是时钟或真实世界的传感设备（例如用户交互设备）。所产生的数据事件由弧所定义的那样被转移到其他节点，在那里对它们进行处理，这些节点然后会产生进一步的数据事件。最后的数据被某个节点接收，此节点不再传播事件。数据流模型对描述时间驱动的数据处理这样操作的系统是十分有用的。一些重要的性质是，由一个节点所产生的数据事件可能被发送到多个目的地。同样地，一个节点可能接收来自多个源的数据事件。



图A-5 抽象数据流模式的例子

重复访问的域

我们知道节点是由域(也可能是其他节点)组成的。域有四种类型: field、exposeField、eventOut、eventIn。一个简单域只能在初始化时设定。eventIn是可以被数据流所设定的域。eventOut是不能被设定的域，它只能读。exposedField既可以在初始化时设定也可以在运行时设定和读取。

我们已经看到过若干类型的域了，例如SFBool和SFVec3f。类型的完全列表包括下面的单值域: SFBool、SFVec2f、SFVec3f、SFRotation、SFFloat、SFNode、SFString、SFColor、SFTime、SFImage和SFInt32，对应的多值域是: MFColor、MFFloat、MFInt32、MFNode、MFRotation、MFString、MFTime、MFVec2f和MFVec3f。 531

下列各例通过节点描述和确定哪些域可以被实时设定和/或读/写说明各种不同域和域的类型的使用。

```
Box {
  field SFVec3f size 2 2 2
}
```

方盒的尺寸是一个简单域，因此它只能在初始时刻设定。

```
Shape {
  exposedField SFNode appearance NULL
  exposedField SFNode geometry NULL
}
```

Shape节点的几何与外观都是exposedField，因此任一个都可以在初始化时设定，也可以在运行时被改变。

```
OrientationInterpolator {
  eventIn SFFloat set_fraction
  eventOut SFRotation value_changed
  exposedField MFFloat key []
  exposedField MFRotation keyValue []
}
```

OrientationInterpolator节点将在“动画、时间传感器和内插器”中讨论。它是一个单浮点数，取值在0和1之间，输出一个旋转值（SFRotation)。域的不同种类暗示当前的浮点值不能被读，而且当前的旋转值不能被直接设定只能被读取。这些值都没有初始值。key和keyValue域定义了浮点值是如何映射到旋转值的。因为这些是暴露域，它们能在初始化时被设定，并且在运行时刻被改变和读取。

数据流和路径

在抽象数据流模型中，数据是从左边向右边流动的。在VRML中，这对应着数据从eventOut域或exposedField到eventIn或exposedField。为了要建立这样的连接我们使用ROUTE语句。

ROUTE语句确定节点和域的配对来建立连接。所连接的节点通过它们的名字被识别，这个名字用DEF给定。参见“组、变换和场景图”对DEF/USE机制的描述。在VRML中，路径的扇入和扇出都是允许的。举例来说，有多重路径可能影响Shape节点的外观；或特别地，OrientationInterpolator可能驱动多个Trsnsform节点的方向。在数据流中的每个事件都是有时间戳的，因为在同一级中的所有事件都有相同的时间戳，数据流的发生是“即时” 532

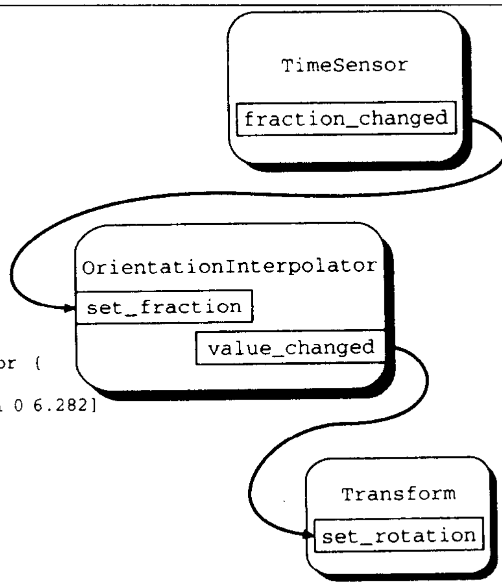
的。事件的处理按照时间戳的顺序，因此从源开始的一级完成之后下一个源才能开始。通过ROUTE机制，我们就可能创建一个循环路径，但是如果我们不允许数据流中的一个事件与先前的事件有相同的时间戳和值的话，这样的循环路径就自然断了。

动画、时间传感器和内插器

数据流机制通过使用时间传感器和内插器生成动画。我们通过图A-6 中的例子来说明，该图给出了例子代码和旋转方盒的图示。在这个例子中，我们通过对TimeSensor的一种基本使用来产生一个周期值。TimeSensor连续地（loop TRUE）生成在0和1之间的浮点值。该值平滑地每隔2秒（cyclInteral 2.0）从0到1改变，在这之后立即复置为0并继续。

533

```
DEF TRANS Transform {
  children [USE ONE_BOX ]
}
DEF TIMER TimeSensor {
  loop TRUE
  cycleInterval 2.0
}
DEF ROTATOR OrientationInterpolator {
  key [0, 0.5, 1]
  keyValue [0 1 0 0, 0 1 0 3.141, 0 1 0 6.282]
}
ROUTE TIMER.fraction_changed TO
  ROTATOR.set_fraction
ROUTE ROTATOR.value_changed TO
  TRANS.rotation
```



图A-6 旋转纹理立方体的例子

OrientationInterpolator像所有的内插器一样，其值在0和1之间。它产生一个方向，此方向将被传递到一个Transform节点。总的效果是产生一个连续旋转的对象。

TimeSensor是一个复合节点，可以用来产生连续或单个的执行循环。TimeSensor可以被所设定的开始时刻和终止时刻所触发。在下一小节中我们将给出一个更复杂的例子。

OrientationInterpolator 是一组以相同方式运行的内插器中的一个。我们用最简单的例子ScalarInterpolator说明内插器的操作，但是这些技术也适用于其他内插器OrientationInterpolator、ColorInterpolator、CoordinateInterpolator、NormalInterpolator和 PositionInterpolator。

ScalarInterpolator的定义如下:

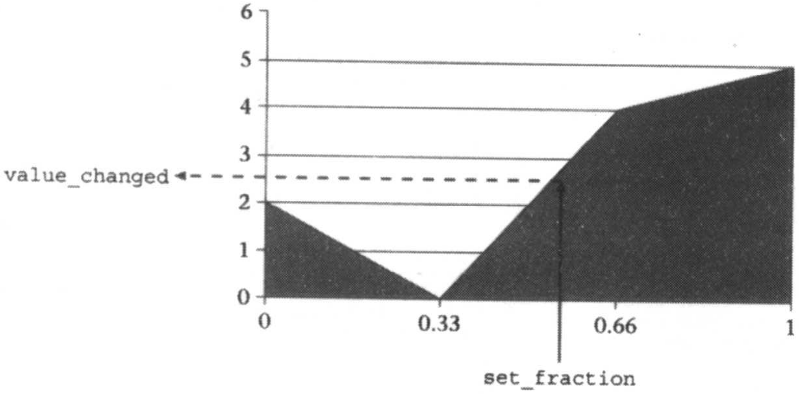
```
ScalarInterpolator {
   eventIn SFFloat set_fraction
   exposedField MFFloat key []
   exposedField MFFloat keyValue []
   eventOut SFFloat value_changed
}
```

exposedFields、key和keyValue包含相同数目的元素，它们定义了一个从set_fraction到value_changed的映射。举例来说：

```
key [0, 0.33, 0.66, 1]
  keyValue [2.0, 0.0, 4.0, 5.0]
```

534

定义的映射用图A-7解释。



图A-7 ScalarInerpolator的key和keyValue的图示

value_changed来自于一个简单的插值，该插值位于限制set_fraction 事件的key的keyValues之间。显然必须保证对每个key只能有一个keyValue，这样这些key是按单调增加顺序定义的。

A.4 基于VRML的交互式体验

脚本构造

传感器和内插器不足以描述复杂的行为。VRML通过Script节点支持行为的脚本。这允许场景制作者创建带有任意域的节点，且能提供所需的功能。

当前 VRML97 支持两种脚本语言：JavaScript 和 Java。两种语言有类似的应用编程接口，且都有三大方面的功能：数据流事件处理、场景图操作以及VRML浏览器界面。

Script节点的主要目的是提供在一连串数据流事件中的复杂数据处理。Script节点在有事件到来的时候确定一组将要响应的输入域。形成这些输入域中每一个域的一连串数据流将引起位于Script中的一个被调用的函数。脚本执行实际机制和名称依赖于脚本语言。举例来说，在JavaScript 中，场景制作者创建一个与输入域具有相同名称的函数。然后，Script节点在输入数据上执行一些任务，并通过写入一个输出域有选择地触发进一步的一连串事件数据流。 535

我们用图A-8和A-9中的例子说明数据流处理行为。图A-8给出了一个母牛模型的VRML代码，该模型可以通过SphereSensor旋转，当它被倒转过来和正过来的时候会发出“哞”的叫声。图A-9 给出了节点之间的对应路径。

Script节点不仅提供一般的数据流处理能力，它们也能创建和删除场景图中的节点，以及创建和删除节点之间的ROUTE。最后，还有一小组功能可以用来在脚本节点中控制和查询浏览器本身。这些包括强制载入一个新场景或查询当前的帧频。关于对这些更进一步的讨论以及对脚本构造更详细的介绍，可以参阅 VRML97比较深入的教程（例如Carey and Bell，1997）。

```
#VRML V2.0 utf8

Group {
   children [
      DEF ROT SphereSensor {
      }
      ,
      DEF TRANS Transform {
         translation 0 0 0
         children [
            Inline {
               url "cow-model.wrl"
         },
         Sound {
         source DEF
            MOO_SOUND
               AudioClip {
                  url "moo.wav"
               }
         }
      ]
   }
   ]
}

ROUTE ROT.rotation_changed
   TO TRANS.set_rotation
```

```
DEF MOOER Script   {
   eventIn SFRotation set_rotation
   eventOut SFTime mooTime

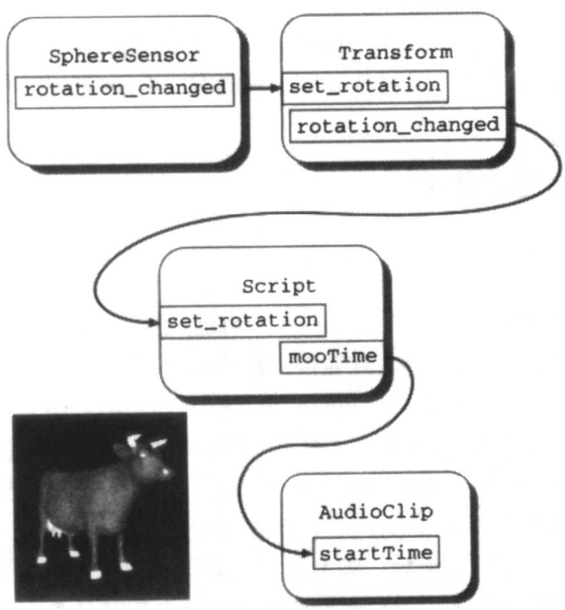
   field SFBool last_updir TRUE
   field SFBool first TRUE
   field SFBool lastup TRUE
   url "javascript:

function set_rotation(value, timestamp){
   updir = value.multVec(new SFVec3f(0,1,0));
   up = (updir.y > 0.0);
   if (first) {
      lastup = up;
      first = false;
   } else {
      if ((!lastup) && up) {
         mooTime = timestamp;
      }
      lastup = up;
   }
}
"
}

ROUTE TRANS.rotation_changed
   TO MOOER.set_rotation
ROUTE MOOER.mooTime TO MOO_SOUND.startTime
```

图A-8 母牛例子的VRML代码

536



图A-9 母牛例子的数据流

A.5 小结

这个附录对VRML作了一个总体介绍，足够读者了解和实验本书中的 VRML例子了。我们没有涉及太多关于VRML多媒体方面的内容（例如电影纹理和立体声），也没有对 VRML的脚本构造给予太多的介绍。我们也没有对有趣的、功能强大的PROTO 机制作任何介绍，该机
537 制允许我们定义新的节点类型。进一步的例子可以在网上许多地方看到，大家可以参考Web3D 协会的主页(http: //www.web3d.org)。

参考文献

Airey, J.M., Rohlf, J.H., and Brooks, Jr., F.P. (1990). Towards image realism with interactive update rates in complex virtual building environments, *Computer Graphics* (1990 Symposium on Interactive 3D Graphics), 24(2), 41–50.

Amanatides, J. (1984) Ray tracing with cones, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 18, 129–135. Addison-Wesley.

Amanatides, J. (1987) A fast voxel traversal algorithm for ray tracing, *Proc. Eurographics 87*, pp. 3–10.

Andreev, R.D. (1989) Algorithm for clipping arbitrary polygons, *Computer Graphics Forum*, 8(3), 183–91.

Appel, A. (1968) Some techniques for shading machine renderings of solids, *Proc. AFIPS JSCC 1968*, 32, 37–45.

Arvo, J. and Kirk, D. (1987) Fast ray tracing by ray classification, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 21(4), 55–64. Addison-Wesley.

Arvo, J. and Kirk, D. (1990) Particle transport and image synthesis, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 24(4), 63–66. Addison-Wesley.

Ascension, MotionStar (`http://www.ascension-tech.com/`)

Aserinsky, E. and Kleitman, N. (1953) Regularly occurring periods of eye motility, and concomitant phenomena, during sleep, *Science*, 118, 273–74.

Atherton, P.R., Weiler, K., and Greenberg, D. (1978) Polygon shadow generation, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, (12), 275–81. Addison-Wesley.

Azuma, R.T. (1997) A survey of augmented reality, *Presence: Teleoperators and Virtual Environments*, 6(4).

Badler, N.I., Hollick, M., and Granieri, J. (1993) Real-time control of a virtual human using minimal sensors, *Presence: Teleoperators and Virtual Environments*, 2(1), 82–6.

Barfield, W. and Hendrix, C. (1995) The effect of update rate on the sense of presence within virtual environments, *Virtual Reality: The Journal of the Virtual Reality Society*, 1(1), 3–16.

Bartels, R.H., Beatty, J.C., and Barsky, B.A. (1987) *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Inc., Los Altos, California.

Baum, D.R., Mann, S., Smith, K.P., and Winget, J.M. (1991) Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In T.W. Sederberg (ed.), *ACM Computer Graphics*, vol. 25, 51–60.

Baumgart, B.G. (1974) Geometric modeling for computer vision. AIM-249, STA–CS-74-463, CS Dept, Stanford U.

Baumgart, B.G. (1975) A polyhedron representation for computer vision, IFIPS.

Bentley, J.L. (1975) Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 18, 509–17.

Bergeron, P. (1986) A general version of Crow's shadow volumes, *IEEE CG&A*,

6(9), 17–28.

Bernardini, F., Klosowski, J.T., and El-Sana, J. (2000) Directional discretized occluders for accelerated occlusion culling, *Computer Graphics Forum*, 19(3).

Bier, E.A. (1990) Snap-dragging in three dimensions, *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, in *Computer Graphics*, 24(2), 193–204.

Bishop, G. and Weimer, D.M. (1986) Fast Phong shading, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 20(4), 103–6. Addison-Wesley.

Bittner, J., Havran, V., and Slavik, P. (1998) Hierarchical visibility culling with occlusion trees. *Proceedings of Computer Graphics International '98*, pp. 207–19.

Blaxxun, http://www.blaxxun.com

Blinn, J.F. (1978) Simulation of wrinkled surfaces, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 12(3), 286–92. Addison-Wesley.

Blinn, J.F. (1988) Jim Blinn's corner: Me and my (fake) shadow, *IEEE Computer*, January, *Graphics and Applications*, 8(1), 82–6.

Blinn, J.F. (1991) A trip down the graphics pipeline – line clipping, *IEEE CG&A*, January, 11(1) 98–105.

Blinn, J.F. and Newell, M.E. (1976) Texture and reflection in computer generated images, *Communications of the ACM*, 19(10), October, 542–47.

Blinn, J.F. and Newell, M.E. (1978) Clipping using homogeneous coordinates, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 12, 245–51. Addison-Wesley.

Bouknight, W.J. (1970) A procedure for generation of three-dimensional half-toned computer graphics presentations, *Communications of the ACM*, 13(9), September, 527–36.

Bouknight, W.J. and Kelley, K. (1970) An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources, *AFIPS Conf. Proc. 36*, 1–10.

Boulic, R., Rezzonico, S., and Thalmann, D. (1996) Multi finger manipulation of virtual objects, *Proc. of ACM Symposium on Virtual Reality Software and Technology VRST'96*, Hong-Kong, July.

Bresenham, J.E. (1965) Algorithm for computer control of digital plotter, *IBM System Journal*, 4(1), 25–30.

Bresenham, J.E. (1977) A linear algorithm for incremental digital display of circular arcs, *Communications of the ACM*, 20(2), 100–6.

Brooks, F.P. Jr (1986) Walkthrough – a dynamic graphics system for buildings, *Proc. 1986 ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October, pp. 9–21.

Brookshire Conner, D., Snibber, S.S., Herndon, K.P., Robbins, D.C., Zeleznik, R.C., and van Dam, A. (1992) Three-dimensional widgets, *Proc. 1992 Symposium on Interactive 3D Graphics*, in *Computer Graphics*, 25(2), pp. 183–88.

Brotman, L.S. and Badler, N.I. (1984) Generating soft shadows with a depth buffer algorithm, *IEEE Computer Graphics & Applications*, 4(10), 71–81.

Bruce, V. and Green, P.R. (1990) *Visual Perception*, 2nd edn, Lawrence Erlbaum Associates, East Sussex UK, ISBN 0-86377-146-7, reprinted in 1992.

Bui-Tong, Phong (1975) Illumination for computer-generated pictures, *Communications of the ACM*, 18(6), 311–17.

Buxton, W. (1986) There's more to interaction than meets the eye: some issues in manual input. In D.A. Norman and S.W. Draper, (eds) *User Centered System Design: Net Perspectives on Human-Computer Interaction*. Lawrence Erlbaum

Associates, Hillsdale, New Jersey, pp. 319–37.

Camahort, E. and Fussell, D. (1999) A geometric study of light field representations. Technical Report TR-99-35, Department Of Computer Science, University of Texas at Austin.

Camahort, E., Lerios, A., and Fussell, D. (1998) Uniformly sampled light fields, *Rendering Techniques '98*, 117–30.

Campbell, A.T. (1991) Modelling global diffuse illumination for image synthesis. PhD Thesis, Department of Computer Science, University of Texas at Austin, December.

Campbell, A.T. and Fussell, D.S. (1990) Adaptive mesh generation for global illumination, *ACM Computer Graphics*, 24(4), 155–64.

Campbell, III, A.T. and Fussell, D.S. (1991) An analytic approach to illumination with area light sources. Technical Report R-91-25, Department of Computer Sciences, University of Texas at Austin.

Carey, R. and Bell, G. (1997) *The Annotated VRML97 Reference*, Addison Wesley Longman, Inc.

Carlbom, I. and Paciorek, J. (1978) Planar geometric projections and viewing transformations, *Computing Surveys*, 10(4), 465–502.

Catmull, E. (1974) A subdivision algorithm for computer display of curved surfaces. PhD Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT.

Chai, J.-X., Tong, X., Chan, S.-C., and Shum, H.-Y. (2000) Plenoptic sampling, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 307–318. Addison-Wesley.

Chan, K.C. and Tan, S.T. (1988) Hierarchical structure to Winged Edge structure: a conversion algorithm, *The Visual Computer*, 4, 133–41.

Chapman, J., Calvert, T.W., and Dill, J. (1991) Spatio-temporal coherence in ray tracing, *Graphics Interface '91*, 101–108.

Chen, E. (1990) Incremental radiosity: an extension of progressive radiosity to an interactive image synthesis system, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 24(4), 135–44. Addison-Wesley.

Chen, M., Mountford, S.J., and Sellen, A. (1988) A study in interactive 3D rotation using 2D control devices, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, August, 121–9, Addison-Wesley.

Chen, S.E. (1995) Quicktime VR – an image-based approach to virtual environment navigation, *Computer Graphics (ACM SIGGRAPH), Annual Conference Series*, 29–38. Addison-Wesley.

Chen, H. and Wang, W. (1996) The feudal priority algorithm on hidden-surface removal, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 55–64. Addison-Wesley.

Chen, S. and Williams, L. (1993) View interpolation for image synthesis. In *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 279–288. Addison-Wesley.

Chen, S.E., Rushmeier, H.E., Miller, G., and Turner, D. (1991) A progressive multi-pass method for global illumination, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 25(4), 165–74. Addison-Wesley.

Chin, N. and Feiner, S. (1989) Near real-time shadow generation using BSP trees, *Computer Graphics*, 23(3), 99–106.

Chin, N. and Feiner, S. (1992) Fast object-precision shadow generation for area light sources using BSP trees. In *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pp. 21–30.

Chrysanthou, Y. (1996) Shadow computation for 3D interaction and animation. PhD thesis, Queen Mary and Westfield College, University of London.

Chrysanthou, Y. (2001) Occlusion culling using tree collapsing. UCL Department of Computer Science, Research Note.

Chrysanthou, Y. and Slater, M. (1992) Dynamic changes to scenes represented as BSP trees. In A. Kilgour and L. Kjelldahl (eds) *Eurographics 92*, September, Blackwells, pp. 321-32.

Chrysanthou, Y. and Slater, M. (1995) Shadow volume BSP trees for fast computation of shadows in dynamic scenes, *Proc. ACM Symposium on Interactive 3D Graphics*, 45-50.

Chrysanthou, Y. and Slater, M. (1997) Incremental updates to scenes illuminated by area light sources. In J. Dorsey and Ph. Slusallek (eds) *Rendering Techniques '97*, pp. 103-14. Springer Computer Science.

Clark, J.H. (1976) Hierarchical geometric models for visible surface algorithms, *Communications of the ACM*, 19(10), 547-54.

Claussen, U. (1989) On reducing the phong shading method. In W. Hansmann, F.R.A. Hopgood and W. Strasser (eds) *Eurographics 89*, pp. 333-44 (North-Holland).

Cleary, J.G. and Wyvill, G. (1988) Analysis of an algorithm for fast ray tracing using uniform space subdivision, *The Visual Computer*, 4, 65-83.

Cohen, M.F. and Greenberg, D.P. (1985) The hemi-cube: a radiosity solution for complex environments, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 19(3), 31-40. Addison-Wesley.

Cohen, M.F., Greenberg, D.P., Immel, D.S., and Brock, P.J. (1986) An efficient radiosity approach for realistic image synthesis, *IEEE Computer Graphics and Applications*, 6(3), 26-35.

Cohen, M.F., Shenchang, Ec., Wallace, J.R., and Greenberg, D.P. (1988) A progressive refinement approach to fast radiosity image generation, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 22(4), 75-84. Addison-Wesley.

Cohen-Or, D., Fibich, G., Halperin, D., and Zadicario, E. (1998) Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes, *Computer Graphics Forum*, 17(3), 243-54.

Cohen-Or, D., Chrysanthou, Y., Silva, C., and Drettakis, G. (2000) Visibility, problems, techniques and applications. *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, course notes. Addison-Wesley.

Cook, R.L., Porter, T., and Carpenter, L. (1984) Distributed ray tracing, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 18, 137-45. Addion-Wesley.

Coorg, S. and Teller, S. (1996) Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pp. 78-87.

Coorg, S. and Teller, S. (April 1997) Real-time occlusion culling for models with large occluders. In *1997 Symposium on Interactive 3D Graphics*, pp. 83-90.

COVEN, The Collaborative Virtual Environment project, `http://coven.lancs.ac.uk`

Coxeter, H.S.M. (1973) *Regular Polytopes*, Dover Publications, New York.

Crocker, G.A. (1984) Invisibility coherence for faster scan-line hidden surface algorithms, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 18(3), 95-102. Addison-Wesley.

Crow, F. (1977) Shadow algorithms for computer graphics, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 11(2), 242-7. Addison-Wesley.

Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R.V., and Hart, J.C. (1992) The CAVE: audio visual experience automatic virtual environment, *Communications of the ACM*, 36(5), June, 65–72, ACM Press.

Cruz-Neira, C., Sandin, D.J., and DeFanti, T.A. (1993) Surround-screen projection-based virtual reality: the design and implementation of the CAVE, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 135–42. Addison-Wesley.

Cyrus, M. and Beck, J. (1978) Generalised two- and three-dimensional clipping, *Computers and Graphics*, 3(1), 23–8.

Darken, R.P., Cockayne, W.R., and Carmein, D. (1997) The omni-directional treadmill, a locomotion device for virtual worlds, *Proceedings of UIST '97*, Banff, Canada, October 14–17, 1997, pp. 213–221, ACM Press.

Darken, R.P., Allard, T., and Achille, L.B. (1999) Spatial orientation and wayfinding in large-scale virtual spaces II: guest editors' introduction to the special issue, *Presence: Teleoperators and Virtual Environments*, 8(6), iii–vi.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997) *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, Heidelberg, New York, ISBN 3-540-61270-X.

de Casteljau, P. (1986) *Shape Mathematics and CAD*, Kogan Page Ltd, London.

Dippe, M. and Swensen, J. (1984) An adaptive subdivision algorithm and parallel architecture for realistic image *Synthesis, Computer Graphics (SIGGRAPH)*, 18(3), 149–58.

Dorr, M. (1990) A new approach to parametric line clipping, *Computers and Graphics*, 14(3/4), 449–64.

Draper, J.V., Kaber, D.B., and Usher, J.M. (1998) Telepresence, *Human Factors*, 40(3), 354–75.

Drettakis, G. (1994) Structured sampling and reconstruction of illumination for image synthesis. PhD thesis, Department of Computer Science, University of Toronto.

Drettakis, G. and Fiume, E. (1994) A fast shadow algorithm for area light sources using backprojection, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 223–30. Addison-Wesley.

Durand, F., Drettakis, G., Thollot, J., and Puech, C. (2000) Conservative visibility preprocessing using extended projections, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 239–48. Addison-Wesley.

Duvanenko, V.J., Robbins, W.E., and Gyurcsik, R.S. (1990) Improving line segment clipping, *Dr Dobb's Journal of Software Tools*, 15(7), July, 36, 38, 40, 42, 44–5, 98, 100.

Edwards, B. (1999) *The New: Drawing on the Right Side of the Brain*, Jeremy P. Tarcher/Putnam, New York.

Ellis, S.R. (1991) Nature and origin of virtual environments: a bibliographic essay, *Computing Systems in Engineering*, 2(4), 321–47.

Ellis, S.R. (ed.) (1993) *Pictorial Communication in Virtual and Real Environments*, 2nd edn, Taylor and Francis Ltd, London, ISBN 0-74840-0082-6.

Ellis, S.R., Young, M.J., Adelstein, B.D., and Ehrlich, S.M. (1999) Discrimination of changes in latency during head movement. In *Proceedings of HCI '99*, Munich, pp. 1129–33.

Ellis, S.R., Adelstein, B.D., and Young, M.J. (2000) Studies and management of latency in virtual environments. In *Proc. 3rd International Conference on Human and Computer*, Aizu Uinversity, pp. 291–300.

Fakespace Inc., Pinch Glove (`http://www.fakespace.com/products/`

pinch.html)

Fallon, N. and Chrysanthou, Y. (2001) Image space occlusion for urban scenes. UCL Department of Computer Science Research Note.

Farin, G.E. (1992) From conics to NURBS: a tutorial and survey, *IEEE CG&A*, September, 78–86.

Farin, G.E. (1996) *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide* (4th edn) (Computer Science and Scientific Computing Series), Academic Press; ISBN: 0122490541.

Feynman, R.F., Leighton, R.B., and Sands, M. (1977) *The Feynmann Lectures on Physics*, Addison-Wesley Publishing Company, Reading, Mass. (sixth printing), ISBN 0-201-02116-1-P.

Foley, J.D., Wallace, V.L., and Chan, P. (1984) The human factors of computer graphics interaction techniques, *IEEE Computer Graphics and Applications*, November, 13–48.

Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F. (1990) *Computer Graphics: Principles and Practice*, 2nd edn, Addison-Wesley Publishing Company.

Freud, S. (1983) *The Interpretation of Dreams*, Avon; ISBN: 0380010003.

Fuchs, H., Kedem, Z.M., and Naylor, B.F. (1980) On visible surface generation by a priori tree structures, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 14(3), 124–33. Addion-Wesley.

Fuchs, H., Abram, G.D., and Grant, E.D. (1983) Near real-time shaded display of rigid objects, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 17(3), 65–72. Addison-Wesley.

Fujimoto, A., Tanaka, T., and Iwata, K. (1986) ARTS: Accelerated Ray-Tracing System, *IEEE CG&A*, 6(4), 16–26.

Fung, K.Y., Nicholl, T.M., and Dewdney, A.K. (1992) A run-length slice line drawing algorithm without division operations, *Eurographics 92, Computer Graphics Forum*, 11(3), Conference Issue, eds A. Kilgour and L. Kjelldahl, 267–277.

Funkhouser, T.A., and Séquin, C.H. (1993) Adaptive display algorithm for interactive frame rates during visualisation of complex virtual environments, Proceedings of SIGGRAPH '93. In *Computer Graphics (ACM SIGGRAPH) Annual Conference Series, 1993*, 247–254. Addison-Wesley.

Garland, M. and Heckbert, P.S. (1997) Surface simplification using quadric error metrics, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 209–216. Addison-Wesley.

Gatenby, N. (1995) Incorporating hierarchical radiosity into discontinuity meshing radiosity. PhD thesis, University of Manchester, Manchester, UK.

Gersho, A., and Gray, R. (1992) *Vector Quantization And Signal Compression*, Kluwer Academic Publishers.

Gibson, J.J. (1986) *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Associations, Publishers, New Jersey.

Gibson, S. and Hubbold, R.J. (1997) Perceptually-driven radiosity, *Computer Graphics Forum*, 16(2), 129–40.

Gigus, Z. and Malik, J. (1990) Computing the aspect graph for the line drawings of polyhedral objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(2), 113–33.

Gigus, Z., Canny, J., and Seidel, R. (1991) Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(6), 542–51.

Glassner, A.S. (1984) Space subdivision for fast ray tracing, *IEEE Computer Graphics and Applications*, 4(10), 15–22.

Glassner, A.S. (ed.) (1989) *An Introduction to Ray Tracing*, Academic Press, San Diego, CA. ISBN 0-12-286160-4.

Glassner, A.S. (1995) *Principles of Digital Image Synthesis*, Vols 1–2, Morgan Kaufmann Publishers, San Francisco, California, ISBN 1-55860-276-3.

Goldman, R.N. (1990) Blossoming and knot insertion algorithms for B-spline curves, *Computer Aided Geometric Design*, 7, 69–81.

Goldsmith, J. and Salmon, J. (1987) Automatic creation of object hierarchy for ray tracing, *IEEE CG&A*, 7(5), 14–20.

Gomes, J. and Velho, L. (1997) *Image Processing for Computer Graphics*, Springer-Verlag, New York, ISBN 0-387-94854-6.

Goral, C., Torrance, K.E., and Greenberg, D. (1984) Modeling the interaction of light between diffuse surfaces, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 18(3), 213–22. Addison-Wesley.

Gordon, D. and Chen, S. (1991) Front-to-back display of BSP trees, *IEEE CG&A*, 11(5), 79–85.

Gortler, S., Grzeszczuk, R., Szeliski, R., and Cohen, M. (1996) The Lumigraph, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 43–52.

Gouraud, H. (1971) Continuous shading of curved surfaces, *IEEE Trans. on Computers*, C(20)-6, 623–629.

Greene, N., Kass, M., and Miller, G. (1993) Hierarchical Z-buffer visibility, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 20, 231–8. Addison-Wesley.

Gregory, R.L. (1998a) *Mirrors in Mind*, Penguin Books, London.

Gregory, R.L. (1998b) *Eye and Brain: The Psychology of Seeing*, 5th edn, Oxford University Press, ISBN 0 19 852412 9.

Gu, X., Gortler, S., and Cohen, M. (1997) Polyhedral geometry and the two-plane parameterization, *Eighth Eurographics Workshop on Rendering*, pp. 1–12.

Haeberli, P. and Akeley, K. (1990) The accumulation buffer: Hardware support for high-quality rendering. *Proc. Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 24(4), 309–18. Addison-Wesley.

Haines, E.A. and Greenberg, D.P. (1986) The Light Buffer: a shadow testing accelerator, *IEEE CG&A*, 6(9), 6–16.

Haines, E. and Wallace, J. (1991) Shaft culling for efficient ray-traced radiosity. In *Proc. Second Eurographics Workshop on Rendering* (Barceloha, Spain, May 1991), Eurographics, Springer Verlag. Also published in SIGGRAPH 91 course notes: Frontiers of Rendering.

Hall, R. (1989) *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, ISBN 3-540-06774-5.

Hall, R. (1999) Comparing spectral color computation methods, *IEEE Computer Graphics & Applications*, 19(4), July/August, 36–45.

H-Anim (1999), Humanoid Animation Specifcation, Version 1.1, `http://ece.uwaterloo.ca/~h-anim/spec1.1/`

Hanrahan, P., Saltzman, D., and Aupperle, L. (1991) A rapid hierarchical radiosity algorithm, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 25(4), 197–206. Addison-Wesley.

Hardt, S. and Teller, S. (1996) High-fidelity radiosity rendering at interactive rates, in *Proc. 7th Eurographics Rendering Workshop*, June, pp. 71–80.

Havran, V. and Bittner, J. (2000) LCTS: Ray shooting using longest common traversal sequences, *Computer Graphics Forum*, 19(3), 59–70 (Proceedings of

EG 2000, Interlaken, Switzerland).

Hedley, D. (1998) Discontinuity meshing for complex environments. PhD thesis, Department of Computer Science, University of Bristol, August.

Heckbert, P.S. (1984) The mathematics of quadric surface rendering and SOID. 3-D Technical Memo No. 4, Three Dimensional Animation Systems Group, Computer Graphics Lab, New York Institute of Technology.

Heckbert, P.S. (1986) A survey of texture mapping, *IEEE CG&A*, 6(11), 56–67.

Heckbert, P.S. (1990) Adaptive radiosity textures for bidirectional ray tracing, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 24, 145–54. Addison-Wesley.

Heckbert, P. (1991) Simulating global illumination using adaptive meshing. PhD thesis, CS Division (EECS), University of California, Berkeley.

Heckbert, P. (1992a) Radiosity in flatland, Eurographics 1992, *Computer Graphics Forum*, 11(3), 181–92.

Heckbert, P. (1992b) Discontinuity meshing for radiosity, Third Eurographics Workshop on Rendering, Bristol, UK, May pp. 203–26.

Heidmann, T. (1991) Real shadows in real time, *IRIS Universe* (18), 28–31.

Held, R.M. and Durlach, N.I. (1992) Telepresence, *Presence: Teleoperators and Virtual Environments*, 1(1), 109–12.

Herf, M. and Heckbert, P. (1996) Fast soft shadows. In Technical Sketches, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, p. 145. Addison-Wesley.

Hodges, L.F. and MacAllister, D.F. (1993) Computing stereo views, in D.F. McAllister (ed.) *Stereo Computer Graphics and Other True 3D Technologies*, Princeton University Press, New Jersey, ISBN 0-691-08741-5.

Hoggar, S.G. (1992) *Mathematics for Computer Graphics*, Cambridge University Press, ISBN 0521 375746.

Hoppe, H. (1996) Progressive meshes. In *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 99–108. Addison-Wesley.

Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang, H. (1997). Accelerated occlusion culling using shadow frusta. In *Proc. 13th International Annual Symposium on Computational Geometry (SCG–97)*, pp. 1–10, ACM Press, New York.

Hutchins, E.L., Hollan, J.D., and Norman, D.A. (1986) Direct manipulation interfaces. In D.A. Norman and S.W. Draper (eds) *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 87–214.

Ihm, I., Park, S., and Lee, R.K. (1997) Rendering Of Spherical Light Fields, *Proc. Pacific Graphics 97*.

Immel, D.S., Cohen, M.F., and Greenberg, D.P. (1986) A radiosity method for non-diffuse environments, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 20(4), 133–42. Addison-Wesley.

Intersense, Intersense IS-900 (`http://www.isense.com/`)

Isaksen, A., McMillan, L., and Gortler, S.J. (2000) Dynamically reparameterized lightfields, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 297–306. Addison-Wesley.

James, A. and Day, A.M. (1998) The priority face determination tree for hidden surface removal, *Computer Graphics Forum*, 17(1), 55–71.

Jansen, F. (1986) Data structures for ray tracing, in L. Kessener, F. Peters and M. van Lierop (eds) ***Data Structures for Raster Graphics, Eurographics Seminar,***

NY, Springer-Verlag, 57–73.

Jenkins, F.A. and White, H.E. (1981) *Fundamentals of Optics*, International Edition, McGraw-Hill, ISBN 0-07-085346-0.

Jensen, H.W. (1996) Global illumination using photon maps, *Rendering Techniques '96, Proc. 7th Eurographics Workshop on Rendering*, pp. 21–30.

Jensen, H.W. and Christensen, N.J. (1995) Photon maps in bidirectional Monte Carlo ray tracing of complex objects, *Computers & Graphics*, 19(2), 215–24.

Jensen, H.W. and Christensen, P.H. (1998) Efficient simulation of light transport in scenes with participating media using photon maps, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 311–20. Addison-Wesley.

Kajiya, J.T. (1986) The rendering equation, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 20(4), 143–50. Addison-Wesley.

Kalawsky, R.S. (1993) *The Science of Virtual Reality and Virtual Environments*, Addison-Wesley.

Kaplan, M.R. (1985) Space-tracing, a constant time ray tracer, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, State of the Art in Image Synthesis notes. Addison-Wesley.

Kaplan, M.R. (1987) The use of spatial coherence, in David F. Rogers and Rac A. Earnshaw (eds), *Ray Tracing Techniques for Computer Graphics*, Springer-Verlag, pp. 184–193.

Kaufman, A.E. (1996) Volume synthesis, in *Discrete Geometry for Computer Imagery*, 6th International Workshop, DGCI'06, Lyon, France, Proceedings, eds S. Miguet, A. Montanvert, S. Ubeda, Springer.

Kay, T.L. and Kajiya, J.T. (1986) Ray tracing complex scenes, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 20(4), 269–78. Addison-Wesley.

Keating, B. and Max, N. (1999) Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques '99 (Proc. of Eurographics Rendering Workshop)*, pp. 197–212, June. Springer Wein, Eurographics.

Kuipers, J.B. (1999) *Quaternians and Rotation Sequences*, Princeton University Press, New Jersey, ISBN 0-691-05872-5.

Kumar, S., Manocha, D., Garrett, W., and Lin, M. (1996) Hierarchical backface computation, *Eurographics Rendering Workshop*, pp. 235–44, Eurographics Springer Wein.

LaBerg, S. (1985) *Lucid Dreaming*, Ballantine Books, NY.

Lee, E.T.Y. (1989) A note on blossoming, *Computer Aided Geometric Design*, (6), 359–62.

Leibovic, K.N. (ed.) (1990) *Science of Vision*, Springer-Verlag, New York, ISBN 0-387-97270-6.

Levoy, M. and Hanrahan, P. (1996) Light field rendering, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 31–42. Addison-Wesley.

Liang, Y.-D. and Barsky, B.A. (1984) A new concept and method for line clipping, *ACM Transactions on Graphics*, 3(1), 1–22.

Liang, Y.-D. and B.A. Barsky (1990) An improved parametric line clipping algorithm, in E.F. Deprettere (ed.), *Algorithms and Parallel VLSI Architectures*, Elsevier Science Publishers, Amsterdam. Conference held 10–16 June 1990 in Pont-à-Mousson, France.

Lischinski, D., Tampieri, F., and Greenberg, D. (1992) Discontinuity meshing for accurate radiosity, *IEEE Computer Graphics and Applications*, 12(6), 25–39.

Loscos, C. and Drettakis, G. (1997) Interactive high-quality soft shadows in

scenes with moving objects, in *Proc. Eurographics '97, Computer Graphics Forum (Conference Issue)*, 17(3), 219–230.

Luebke, D. and Georges, C. (1995) Portals and mirrors: Simple, fast evaluation of potentially visible sets. In P. Hanrahan and J. Winget (eds), *1995 Symposium on Interactive 3D Graphics*, pp. 105–6, ACM SIGGRAPH.

Maciel, P.W.C. and Shirley, P. (1995) Visual navigation of large environments using textured clusters, Symposium on Interactive 3D Techniques, *Proc. 1995 symposium on Interactive 3D graphics*, April 9–12, Monterey, CA, USA, pp. 95–102.

Mackinlay, J., Card, S.K., and Robertson, G.G. (1990) A semantic analysis of the design space of input devices, *Human–Computer Interaction*, Vol. 5, pp. 145–90, Lawrence Erlbaum Associates.

McAllister, D.F. (1993) *Stereo Computer Graphics and Other True 3d Technologies*, Princeton University Press.

McCool, M.D. (2000) Shadow volume reconstruction from depth maps, *ACM Transactions on Graphics*, 19(1), 1–26.

McMillan, L. and Bishop, G. (1995) Plenoptic modeling: an image-based rendering system, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, Los Angeles, CA, August 6–11, 39–46.

Melzer, J.E. and Moffitt, K. (1996) *Head-Mounted Displays: Designing for the User*, McGraw-Hill.

Meyer, K., Applewhite, H.L., and Biocca, F.A. (1992) A survey of position trackers, *Presence: Teleoperators and Virtual Environments*, 1(2), 173–200.

Mine, M., Brooks, F.P. Jr, and Séquin, C. (1997) Moving objects in space: exploiting proprioception in virtual environment interaction, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, Los Angeles, CA, ACM Press, 19–26. Addison-Wesley.

Minsky, M. (1980) Telepresence, *Omni*, June, 45–51.

Möller, T. (1997) A fast triangle–triangle intersection test, *Journal of Graphics Tools*, 2(2), 25–30.

Mortenson, M.E. (1985) *Geometric Modeling*, John Wiley & Sons.

Muller, H. and Winckler, J. (1992) Distributed image synthesis with breadth-first ray tracing and the ray-z-buffer, data structures and efficient algorithms. Final Report on the DFG Special Initiative, in B. Monien and T. Ottmann (eds), *Lecture Notes in Computer Science*, 594, 124–47, Springer-Verlag.

Nakamaru, K. and Ohno, Y. (1997) Breadth-first ray tracing utilizing uniform spatial subdivision, *IEEE Transactions on Visualization and Computer Graphics*, 3(4), 316–28.

Naylor, B. (1990) SCULPT: an interactive solid modeling tool, *Graphics Interface 90*, Morgan-Kaufmann Publishers, pp. 138–55.

Naylor, B.F. (1992) Partitioning tree image representation and generation from 3D geometric models. In *Proc. Graphics Interface '92*, pp. 201–12, Canadian Information Processing Society.

Naylor, B.F. (1993) Constructing good partitioning trees, in *Proc. Graphics Interface '93*, pp. 181–91.

Naylor, B., Amantides, J., and Thibault, W. (1990) Merging BSP trees yields polyhedral set operations, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 24(4), 115–24. Addison-Wesley.

Neider, J., Davis, T., and Woo, M. (1993) *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1*, Addison-Wesley Publishing Company.

Newell, M.E., Newell, R.G., and Sancha, T.L. (1972) A solution to the hidden surface problem, *Proc. ACM National Conference*, pp. 443–50, ACM.

Newman, W.M. and Sproull, R.F. (1979) *Principles of Interactive Computer Graphics*, 2nd edn, McGraw-Hill.

Nicholl, R.A. and Nicholl, T.N. (1990) Performing geometric transformations by program transformation, *ACM Transactions on Graphics*, 9(1), 28–40.

Nicholl, T.M., Lee, D.T., and Nicholl, R.A. (1987) An efficient new algorithm for 2-D line clipping: its development and analysis, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 21(4), 253–62. Addison-Wesley.

Nielsen, G.M. and Olsen, D.R. Jr (1986) Direct manipulation for 3D objects using 2D locator devices, *Proc. 1986 Workshop on Interactive 3D Graphics*, Chapel Hill, NC, USA, October 23–24, pp. 175–82, ACM Press.

Nishita, T. and Nakamae, E. (1983) Half-tone representation of 3-D objects illuminated by area sources or polyhedron sources, *Proc. COMPSAC 83: The IEEE Computer Society's Seventh Internat. Computer Software and Applications Conf.*, pp. 237–42, IEEE.

Ohta, M. and Maekawa, M. (1987) Ray coherence theorem and constant time ray tracing algorithm, *Computer Graphics 1987* (Proc. CG International '87), pp. 303–314, Springer Verlag.

Oliveira, J. and Buxton, B. (2001) Lightweight virtual humans, *Proc. EUROGRAPHICS-UK 2001*, University College London, March.

OpenGL Achitecture Review Board (1992) *OpenGL Reference Manuel: The Official Reference Document for OpenGL, Release 1*, Addison-Wesley Publishing Company.

Paterson, M.S. and Yao, F.F. (1989) Binary partitions with applications to hidden surface removal and solid modeling, in *Proc. 5th Annual ACM Symposium on Computational Geometry*, pp. 23–32, ACM.

Paterson, M.S. and Yao, F.F. (1990) Optimal binary space partitions for orthogonal objects, *Discrete Computational Geometry*, (5), 485–503.

Perlin, K., Paxia, S., and Kollin, J.S. (2000) An autostereoscopic display, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 319–326. Addison-Wesley.

Pertaub, D.-P., Slater, M., and Barker, C. (2001) An experiment on public speaking anxiety in response to three different types of virtual audience, *Presence: Teleoperators and Virtual Environments*, MIT Press, in press.

Phillips, C.B., Badler, N.I., and Granieri, J. (1992) Automatic viewing control for 3D direct manipulation, Proc. 1992 Symposium on Interactive 3D Graphics, in *Computer Graphics*, 25(2), 71–4.

Pineda, J. (1988) A parallel algorithm for polygon rasterization, *Computer Graphics*, 22(4), 17–20.

Pitteway, M.L.V. (1967) Algorithm for drawing ellipses or hyperbolae with a digital plotter, *Computer Journal*, 10(3), 282–289.

Polhemus Inc. 3SPACE Fastrak and 3BALL (`http://www.polhemus.com/`)

Popescu, V., Eyles, J., Lastra, A., Steinhurst, J., England, N. and Nyland, L. (2000) The WarpEngine: an architecture for the post-polygonal age, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 433–42. Addison-Wesley.

Preparata, F.P. and Shamos, M.I. (1985) *Computational Geometry: An Introduction*, Texts and Monographs in Computer Science, Springer, New York.

Puerta, A.M. (1989) The power of shadows: shadow stereopsis, *Journal of the Optical Society of America*, 6, 309–11.

Ramshaw, L. (1987a) Blossoming: a connect the dots approach to splines, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

Ramshaw, L. (1987b) Beziers and B-splines as multiaffine maps, *Theoretical Foundations of Computer Graphics and CAD, Proc. NATO International Advanced Study Institute*, ed. R.A. Earnshaw, pp. 757–76, Springer-Verlag.

Ramshaw, L. (1989) Blossoms are polar forms, *Computer Aided Geometric Design*, 6, 323–358.

Rappaport, A. (1991a) An efficient algorithm for line and polygon clipping, *The Visual Computer*, 7(1), 19–28.

Rappaport, A. (1991b) Rendering curves and surfaces with hybrid subdivision and forward differencing, *ACM Transactions on Graphics*, 10(4), October, 323–341.

Reeves, W.T., Salesin, D.H., and Cook, R.L. (1987) Rendering antialiased shadows with depth maps, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 21, 283–91. Addison-Wesley.

Robinett, W. and Holloway, R. (1992) Implementation of flying, scaling and grabbing in virtual worlds, *Proc. 1992 Symposium on Interactive 3D Graphics*, March 29–April 1, Cambridge, MA, ACM, pp. 189–192.

Robinett, W. and Rolland, J.P. (1992) A computational model for the stereoscopic optic of a head-mounted display, *Presence: Teleoperators and Virtual Environments*, 1(1), 45–62.

Rokne, J.G. and Rao, Y. (1992) Double-step incremental linear interpolation, *ACM Transactions on Graphics*, 11(2), 183–92.

Rokne, J.G. and Wyvill, B. (1990) Fast line scan-conversion, *ACM Transactions on Graphics*, 9(4), 376–88.

Rohlf, J. and Helman, J. (1994) IRIS Performer: a high performance multiprocessing toolkit for real-time 3D graphics, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 381–394. Addison-Wesley.

Roth, S.D. (1982) Ray casting for modeling solids, *Computer Graphics and Image Processing*, 18, 109–44.

Rothbaum, B.O., Hodges, L.F., Kooper, R., Opdyke, D., Williford, J., and North, M.M. (1995) Effectiveness of computer-generated (virtual reality) graded exposure in the treatment of acrophobia, *American Journal of Psychiatry*, 152, 626–8.

Sacks, O.W. (1998) *The Man Who Mistook His Wife for a Hat: And Other Clinical Tales*, Touchstone Books, ISBN: 0684853949.

Sadagic, A. and Slater, M. (2000) Dynamic polygon visibility ordering for head-slaved viewing in virtual environments, *Computer Graphics Forum*, 19(2), 111–22.

Salesin, D. and Stofi, J. (1990) Rendering CSG models with a ZZ-Buffer, *Computer Graphics*, 24(4), 67–76.

Samet, H. (1990) *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA.

Schaufler, G. (1995) Dynamically generated impostors, in D.W. Fellner (ed.) *GI Workshop: Modeling – Virtual Worlds – Distributed Graphics*, infix Verlag, November, 129–35.

Schaufler, G. (1996) Exploiting frame to frame coherence in a virtual reality system, *VRAIS '96*, Santa Clara, California, April 1996, pp. 95–102.

Schaufler, G., Dorsey, J., Decoret, X., and Sillion, F.X. (2000) Conservative volumetric visibility with occluder fusion, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 229–38. Addison-Wesley, ISBN 1-58113-208-5.

Schumacker, R., Brand, B., Gilliland, M., and Sharp, W. (1969) Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air

Force Systems Command, Brooks AFB, TX.

Scott, N., Olsen, D. and Gannet, E. (1998) An Overview of the VISUALIZE fx Graphics Accelerator Hardware, *The Hewlett-Packard Journal*, May, 28–34.

Segal, M., Korobkin, C., Van Widenfelt, R., Foran, J., and Haeberli, P. (1992) Fast shadow and lighting effects using texture mapping, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 26(2), 249–52. Addison-Wesley.

Seidel, H.-P. (1989) A new multiaffine approach to B-splines, *Computer Aided Geometric Design*, 6, 23–32.

Seitz, S. and Kutulakos, K. (1998) Plenoptic image editing, in *Proc. Sixth Int. Conf. on Computer Vision*, pp. 17–24.

Shade, J., Lischinski, D., Salesin, D.H., DeRose, T. and Snyder, J. (1996) Hierarchical image caching for accelerated walkthroughs of complex environments, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 75–82. Addison-Wesley.

Shi, K.K., Edwards J.A., and Cooper D.C. (1990) An efficient line clipping algorithm, *Computers and Graphics*, 14(2), 297–301.

Shirley, P. (2000) *Realistic Ray Tracing*, A.K. Peters Ltd, ISBN: 1568811101.

Shirley, P., Wade, B., Zareski, D., Hubbard, P., Walter, B., and Greenberg, D.P. (1995) Global illumination via density estimation, *Proc. Sixth Eurographics Workshop on Rendering*, June, pp. 187–99.

Shue-Ling Lien, Shantz, M., and Pratt, V. (1987) Adaptive forward differencing for rendering curves and surfaces, *Computer Graphics*, 21(4), 111–17.

Singhal, S. and Zyda, M. (1999) *Networked Virtual Environments: Design and Implementation*, Addison-Wesley Publishing Co., ISBN: 0201325578.

Skala, V. (1989) Algorithms for 2D line clipping, *New Advances in Computer Graphics, Proc. of CG International 89*, pp. 121–8, eds R.A. Earnshaw and B. Wyvill, Springer-Verlag, Tokyo, Japan.

Slater, M. (1992a) A comparison of three shadow volume algorithms, *The Visual Computer*, 9(1), 25–38.

Slater, M. (1992b) Tracing a ray through uniformly subdivided n-dimensional space, *The Visual Computer*, 9(1), 39–46.

Slater, M. and Barsky, B. (1994) 2D line and polygon clipping based on space subdivision, *The Visual Computer*, 10(7), 407–22.

Slater, M. and Chrysanthou, Y. (1996) View volume culling using a probabilistic cashing scheme, in S. Wilbur and M. Bergamasco (eds), *Proceedings of Framework for Immersive Virtual Environments (FIVE)*, ACM Symposium on Virtual Reality Software and Technology, ACM Press.

Slater, M. and Steed, A. (2000) A virtual presence counter, *Presence: Teleoperators and Virtual Environments*, 9(5), 413–34.

Slater, M. and Usoh, M. (1994) Body centered interaction in immersive virtual environments, in M. Magnenat-Thalman and D. Thalman (eds), *Virtual Reality and Arti-ficial Life*, John Wiley.

Slater, M. and Wilbur, S. (1997) A Framework for Immersive Virtual Environments (FIVE): speculations on the role of presence in virtual environments, *Presence: Teleoperators and Virtual Environments*, 6(6), 603–16.

Sloan, P.-P., Cohen, M.F., and Gortler, S.J. (1997) Time critical lumigraph rendering, *ACM Symposium on Interactive 3D Graphics*, pp. 17–23, ACM SIGGRAPH.

Sobkow, M.S., Pospilsil, P., and Yang, Y.-H. (1987) A fast two-dimensional line

clipping algorithm via line encoding, *Computers and Graphics*, 11(4), 459–67.

Soler, C. and Sillion, F.X. (1998) Fast calculation of soft shadow textures using convolution, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 321–32.

Sony, Glasstron (http://www.sony.com)

Stark, L.W. (1995) How virtual reality works: The illusions of vision in "real" and virtual environments, *SPIE Proceedings: Symposium on Electronic Imaging: Science and Technology*, Feb. 5–10, San Jose, California.

Stark, L.W. and Choi, Y.S. (1996) Experimental metaphysics: the scanpath as an epistemological mechanism, in W.H. Zangemeister, H.S. Stiehl and C. Freksa (eds), *Visual Attention and Cognition*, Elsevier Science, B.V., Ch. 2.

Stark, L.W., Privitera, C.M., Yang, H., Azzariti, M., Ho, Y.F., Blackmon, T., and Chernyak, D. (2001) *Representation of Human Vision in the Brain: How Does Human Perception Recognise Images? Journal of Electronic Imaging (Special Issue on Human Vision)*, 10(1), 123–51.

Steed, A.J., Slater, M., Sadagic, A., Tromp, J., and Bullock, A. (1999) Leadership and collaboration in virtual environments, *IEEE Virtual Reality*, Houston, March, IEEE Computer Society, ISBN 0-7695-0093-5, pp. 112–15.

Stewart, A.J. and Ghali, S. (1994) Fast computation of shadow boundaries using spatial coherence and backprojections. In A. Glassner (ed.) *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 231–38. Addison-Wesley.

Stiles, W.S. and Burch, J.M. (1955) Interim report to the Commission Internationale de l'Éclairage Zurich, 1955, on the National Physical Laboratory's investigation of colour-matching (1955) with an appendix by W.S. Stiles and J.M. Burch. *Optica Acta*, 2, 168–81.

Stiles, W.S. and Burch, J.M. (1959) NPL colour-matching investigation: Final report. *Optica Acta*, 6, 1–26.

Stiles, R., Tewari, S., and Mehta, M. (1997) Adapting VMRL2.0 for Immersive User, *Proc. VRML97 Symposium*, February 24–28, Monterey, CA, ACM SIGGRAPH.

Stockman, A. and Sharpe, L.T. (2000) Spectral sensitivities of the middle- and long-wavelength sensitive cones derived from measurements in observers of known genotype, *Vision Research*, 40, 1711–37.

Sutherland, I.E. (1963) Sketchpad: a man–machine graphical communication system, *Proc. pring Joint Computer Conference*, Spartan Books, Baltimore.

Sutherland, I.E. (1965) A head-mounted three-dimensional display, *AFIPS Conference Proceedings*, Vol. 33, Part I, 1968, pp. 757–64.

Sutherland, I.E. and Hodgman, G.W. (1974) Reentrant polygon clipping, *Communications of the ACM*, 17(1), 32–42.

Sutherland, I.E., Sproull, R.F., and Schumacher, R.A. (1974) A characterization of ten hidden-surface algorithms, *ACM Computing Surveys*, 6(1), 1–55.

Tampieri, F. (1993) Discontinuity meshing for radiosity image synthesis. Ph.D. thesis, Cornell University, Ithaca, NY.

Tecchia, F. and Chrysanthou, Y. (2000) Real-time rendering of densely populated urban environments, Eurographics Rendering Workshop 2000, Brno, Czech Republic, pp. 83–8, Eurographics.

Teller, S.J. and Séquin, C.H. (1991) Visibility preprocessing for interactive walkthroughs, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 25(4), 61–9. Addison-Wesley.

Teller, S.J. (1992) Computing the antipenumbra of an area light source. In E.E. Catmull (ed.) *ACM Computer Graphics*, vol. 26, pp. 139–48.

Teller, S., Bala, K., and Dorsey, J. (1996) Conservative radiance interpolants for ray tracing, *Rendering Techniques '96, Proceedings of the Eurographics Workshop* in Porto, Portugal, June 17–19, X. Pueyo and P. Schroder (eds), Springer, pp. 257–68.

Thomas, F. and Johnston, O. (1981) *Disney Animation: The Illusion of Life*, Aberville Press Publishers, New York.

Thibault, W.C., and Naylor, B.F. (1987) Set operations on polyhedra using binary space partition trees, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 21(4), 153–62. Addison-Wesley.

Tilove, R.B. (1981) Line/polygon classification: a study of the complexity of geometrical classication, *IEEE Computer Graphics and Applications*, April, 75–86.

Torres, E. (1990) Optimization of the binary space partition algorithm (BSP) for the visualization of dynamic scenes, C.E. Vandoni and D.A. Duce (eds), *Eurographics 90*, Elsevier Science Publishers B.V., North-Holland, pp. 507–18.

Usoh, M., Arthur, K., Whitton, M., Bastos, R., Steed, A.J., Slater, M., and Brooks, F. (1999) Walking > Walking-in-Place > Flying, in *Virtual Environments, Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 359–364. Addison-Wesley.

Usoh, M., Catena, E., Arman, S., and Slater, M. (2000) Presence questionnaires in reality, *Presence: Teleoperators and Virtual Environments*, 497–503.

VRML (1997) ISO/IEC (1997) ISO/IEC 14772 Virtual Reality Modeling Language (VRML97).

Wald, I., Slusallek, P. and Benthin, C. (2001) Interactive Distributed Ray Tracing of Highly Complex Models, Rendering Techniques 2001 *Proceedings of the 12th Eurographics Workshop on Rendering*, eds S.J. Gortler and K. Kyszkowski, London 25–27 June 2001, 274–85.

Wallace, J.R., Cohen, M.F., and Greenberg, D.P. (1987) A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 311–20. Addison-Wesley.

Wallace, J.R., Elmquist, K.A., and Haines, E. (1989) A ray tracing algorithm for progressive radiosity, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 23(3), 315–23. Addison-Wesley.

Walter, B., Hubbard, P.M., Shirley, P., and Greenberg, D.P. (1997) Global illumination using local linear density estimation, *ACM Transactions on Graphics*, 16(3), 217–59.

Ward, G.J., Rubinstein, F.M., and Clear, R.D. (1988) A ray tracing solution for diffuse interreflection, *Proc. 15th annual conference on Computer graphics*, ACM, August 1–5, Atlanta, GA, USA, pp. 85–92.

Ward, G.J. and Heckbert, P.S. (1992) Irradiance gradients, in A. Chalmers and D. Paddon (eds), *Third EUROGRAPHICS Workshop on Rendering*, Bristol, pp. 85–98.

Ward-Larson, G. and Shakespeare, R. (1997) *Rendering with Radiance: The Art and Science of Lighting Visualization*, Morgan Kaufmann Publishers, San Francisco, CA, ISBN-1-55860-499-5.

Ware, C. and Arthur, K. *et al.* (1993) Fish tank virtual reality, *Proc. Inter-CHI 93 Conference on Human Factors in Computing Systems*, pp. 37–42.

Ware, C. and Jessome, D.R. (1988) Using the bat: a six-dimensional mouse for object placement, *IEEE Computer Graphics and Applications*, November, 8, 65–70.

Ware, C. and Osborne, S. (1990) Exploration and virtual camera control in virtual three dimensional environments. *Proc. 1990 ACM Symposium on Inter-*

active 3D Graphics, ACM Press.

Ware, C. and Slipp, L. (1991) Using velocity control to navigate 3D graphical interfaces: a comparison of three interfaces, *Proc. Human Factors Society 35th Annual Meeting*, San Francisco, September, pp. 300–4. Human Factors Society.

Warnock, J.E. (1969) A hidden-surface algorithm for computer generated half-tone pictures. University of Utah Computer Science Department, TR 4–15, NTIS AD-753 671.

Watkins, G.S. (1970) A real-time visible surface algorithm. PhD Thesis, University of Utah Computer Science Department Technical Report, UTEC-CSC-7-101.

Watt, A.H. and Watt, M. (1992) *Advanced Animation and Rendering Techniques: Theory and Practice*, Addison-Wesley Publishing Co., ISBN: 0201544121.

Weiler, K. (1980) Polygon comparison using a graph representation, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 14(4), 10–18. Addison-Wesley.

Weiler, K. and Atherton, P. (1977) Hidden surface removal using polygon area sorting, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 11(2), 214–22. Addison-Wesley.

Welch, G., Bishop, G., Vicci, L., Brumback, S., Keller, K., and Colucci, D. (2001) High-performance wide-area optical tracking – the HiBall tracking system, *Presence: Teleoperators and Virtual Environments*, 10(1), in press.

Wenzel, E.M. (1992) Localization in virtual acoustic displays, *Presence: Teleoperators and Virtual Environments*, 1(1), 80–102.

Wernecke, J. (1994) *The Inventor Mentor: Programming Object-Oriented 3d Graphics with Open Inventor*, Addison Wesley.

Whitted, T. (1980) An improved illumination model for shaded display, *Communications of the ACM*, 23(6), 343–9.

Williams, L. (1978) Casting curved shadows on curved surfaces, *Computer Graphics*, 12, 270–4.

Witmer, B.G. and Singer, M.J. (1998) Measuring presence in virtual environments: a presence questionnaire, *Presence: Teleoperators and Virtual Environments*, 7(3), 225–40.

Wonka, P. and Schmalstieg, D. (1999) Occluder shadows for fast walkthroughs of urban environments. In H.-P. Seidel and S. Coquillart (eds) *Computer Graphics Forum*, vol. 18, Eurographics Association and Blackwell Publishers Ltd, pp. C51–C60.

Woo, A., Poulin, P., and Fourier, A. (1990) A survey of shadow algorithms, *IEEE CG&A*, 10(6), 13–31.

Woo, M., Neider, J., Davis, T., and Shreiner, D. (1999) *OpenGL Programming Guide*, 3rd edn, Addison-Wesley.

Wood, D.N., Azuma, D.I, Aldinger, K., Curless, B., Duchamp, T., Salesin, D.H., and Stuetzle, W. (2000) Surface light fields for 3D photography, *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, 287–96. Addison-Wesley.

Worrall, A., Willis, C., and Paddon, D. (1995) Dynamic discontinuities for radiosity. In *Edugraphics + Compugraphics Proceedings*, pp. 367–75, P.O. Box 4076, Massama, 2745 Queluz, Portugal, December.

Worrall, A., Hedley, D., and Paddon, D. (1998) Interactive animation of soft shadows, in *Proc. Computer Animation 1998*, IEEE Computer Society, June, pp. 88–94.

Wylie, C., Romney, G.W., Evans, D.C., and Erdahl, A.C. (1967) *Halftone Perspective Drawings by Computer*, FJCC 67, Thompson Books, Washington DC,

pp. 49–58.

Yagel, R., Cohen, D., and Kaufman, A. (1992) Discrete ray tracing, *IEEE Computer Graphics and Applications*, 12(5), 19–28.

Zhang, H., Manocha, D., Hudson, T., and Hoff III, K.E. (1997) Visibility culling using hierarchical occlusion maps, in T. Whitted (ed.), *Computer Graphics (ACM SIGGRAPH) Annual Conference Series*, pp. 77–88, Addison-Wesley.

Zimmerman, T.G., Lanier, J., Blanchard, C., Bryson, S. and Harvill, Y. (1987) A hand gesture interface device, *Proc. CHI+GI'87*, pp. 189–92, ACM Press.

索　引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

absorption of light （光的吸收），75
abstract data flow in VRML97 （在VRML97中的抽象数据流），531
accommodation （调节），36
accumulation buffer（集聚缓冲区），520
Active Edge Table（活动边表），263-5，271
actors（参与者），5
adaptive forward difference evaluation of polynomials（多项式的自适应前向差分估计），430-3
adaptive radiosity texture (rex) （自适应辐射度纹理(rex)），483
affine maps in CAGD （CAGD中的仿射映射），385-6
affine transformations （仿射变换），60，62
　matrix representation of （矩阵表示），62-4
aliasing （走样），129-30，366
　in CRT displays（在CRT显示器中），109
　in global illumination （在全局光照中），470-2
ambient reflection （环境反射），139
Anchor （Anchor），463
angles and directions （角度和方向），55-9
anisotropic objects （各向异性对象），78
anti-aliasing in real-time rendering （实时渲染中的反走样），520-2
approximate visibility algorithms （近似可见性算法），490
arbitrary polygon（任意多边形），261
architectural scenes （建筑物场景），500-3
area light sources（面光源），298
Ascension MotionStar （Ascension MotionStar），441
aspect graphs（特征图），314-15
augmented reality systems（增强现实系统），437
avatars（化身），5，7

B

B-spline curves （B样条曲线），407-15
　and Bézier points （Bézier 点），410-11
　computation（计算），411-12
　control points（控制点），412-13
　cubic interpolation （三次插值），425-8
　　solution for（解），425-7
　definition（定义），409-10
　functions （函数），415-18
　knot insertion （节点插入），413-14
　knot sequences（节点序列），412-13
　multiple knot insertion（多节点插入），414-15
　piecewise （分段），407
　single polynomial segment（单多项式片段），407-8
　two-polynomial segment （二多项式片段），408-10
back-face elimination （背面删除），244-6
back projection （反向投影），323
backface culling （背面消除），491-3
Barycentric combinations （重心组合），59-61
behavioral realism （行为真实感），11-12
Bernstein basis （Bernstein 基）
　and Bézier curves in CAGD （CAGD 中的 Bézier 曲线），395
　for surfaces in CAGD （CAGD中的表面），419-20
Bézier curves in CAGD （CAGD 中的 Bézier 曲线），394-7
　application to （应用到），398-9
　and B-spline curves（B样条曲线），410-11
　and Bernstein basis （Bernstein 基），395
　in blossoming （在开花中），390-2
　converting from general form （从一般形式中转换），402
　degree raising （升阶），397-9
　polynomial form （多项式形式），394-5
　rational curves （有理曲线），400-2
　tangent vectors （切向量），395-7
Bézier patches in CADG （CAGD 中的 Bézier 面片），
　blossoms for （开花），421-2
　triangular （三角形的），422-5
bi-linear interpolation （双线性插值），270
bidirectional ray tracing （双向光线跟踪），483，484
bidirectional reflectance distribution function（BRDF）（双向反射分布函数（BRDF）），
　and ray tracing （光线跟踪），142，144，466
　reflectance （反射），80，81，82

billboards （布告板），508
binary space partition trees （二叉空间分割树），250-9
constructing （构造），250-5
in dynamic scenes （在动态场景中），255-9
in ray tracing （在光线跟踪中），349-50
rendering （渲染），253
shadow volume (SVBSP) （阴影体 (SVBSP)），299，301，304-9
binocular disparity （双眼视差），36
blossoms in CAGD （在 CAGD 中的开花），384，385，387-8
and Bézier curves （Bézier 曲线），390-2
and continuity （连续性），405 -6
de Casteljau triangles （de Casteljau 三角形），388-90，392-3
and degenerate polynomials （退化多项式），398
for rectangular Bézier patches （矩形 Bézier 面片），421-2
body-centered navigation in VEs （在虚拟环境中以身体为中心的导航），444，454
body simulation in virtual environments （虚拟环境中的人体仿真），439-41
model、building （模型、建立），439-40
tracking （跟踪），440-1
Boehm's knot insertion formula （Boehm 节点插入公式），414
boundaries （边界），230
boundary representation of polyhedra （多面体的边界表示），168
bounding box range tests （包围盒范围测试），446
bounding slab method （包围板方法），346
bounding volumes in ray tracing （光线跟踪中的包围体），150，344-5
hierarchical （层次化的），345-6
selection of （选择），346-7
Bresenham's algorithm （Bresenham 算法），367-70，468
BSP trees （BSP 树），
in dynamic scenes （在动态场景中），255-9
non-uniform space subdivision （非一致空间细分），349-50
umbras （本影），304-9

C

camera （照相机），
composite matrix （合成矩阵），202-3
3D stereo views，creating （三维立体视图，创造），217-24
setting up （建立），217-19
full specification （完整描述），191-4
generalizing （泛化），155-61
mapping from WC to UVN coordinates （从WC到UVN坐标映射），157-9
ray tracing （光线跟踪），159
in VRML97 （在 VRML97中），160
projection （投影），194-202
canonical frames （规范框架），197-200
canonical projection space （规范投影空间），200-1
clipping front and back planes （裁剪前平面和后平面），193-4，201
parallel （平行），195-6
perspective （透视），196-7
plane distances，transforming （平面距离，转换），201-2
type of （类型），192
view implementation and scene graph （视图实现和场景图），209-10
view matrix T，computing （视图矩阵 T，计算），203-9
viewing in OpenGL （在OpenGL 中观察），210-17
canonical frames （规范框架），197-200
matrix derivation （矩阵推导），224-5
canonical projection space （规范投影空间），200-1
canonical viewing space （规范观察空间），236-8
caricatures （漫画），12-14
cathode ray tube (CRT) displays （阴极射线管 (CRT) 显示器），108-11
caustics photon map （焦散光子图），486
caustics radiance （焦散光亮度），488
CAVE concept （CAVE概念），7，436-7
cell-to-cell visibility （单元到单元可见性），501
cells and portals occlusion （单元和入口遮挡），500
center of projection (COP) （投影中心 (COP)），191-2，197-8
clipping polygons （裁剪多边形），237，240
in painting metaphor （在绘画隐喻中），122，128
ray tracing （光线跟踪），465-6
visibility determination （可见性确定），249，253
chromatic light （彩色光），92
chrominance （色度），105
CIE-RGB chromaticity space （CIE-RGB色度空间），101-5
CIE-RGB color matching functions （CIE-RGB 颜色匹配函数），97-101
CIE-XYZ chromaticity space （CIE-XYZ色度空间），105-8
clamping (color clipping) （夹（颜色裁剪）），115

clipping（裁剪），128
front and back planes（前平面和后平面），193-4
incorporating（合并），201
line segments（线段），355-65
Bresenham's algorithm（Bresenham 算法），367-70
Cohen-Sutherland algorithm（Cohen-Sutherland 算法），358-62
2D region（二维区域），357-8
Liang-Barsky algorithm（Liang-Barsky 算法），362-4
Nicholl-Lee-Nicholl clipping（Nicholl-Lee-Nicholl 裁剪），364-5
parametric clipping（参数化裁剪），362-4
rasterization of（光栅化），366-70
ray tracing（光线跟踪），370-4
lines（直线），354-74
polygons（多边形），229-42
in canonical viewing space（在规范观察空间中），236-8
in 3D（在三维中），234-41
in homogeneous space（在齐次空间中），238-41
in projection space（在投影空间中），235-6
Sutherland-Hodgman algorithm (2D)（Sutherland-Hodgman算法（二维）），230-2、234-5
Weiler-Atherton algorithm（Weiler-Atherton 算法），232-4
co-domain（值域），385
Cohen-Sutherland algorithm（Cohen-Sutherland 算法），358-62
coherence，exploiting（相关性，利用），262
collision detection（碰撞检测），435、444
general（一般的），446-7
object pair（对象对），444-6
Collision sensor（碰撞传感器），463
color（颜色），
CIE-RGB chromaticity space（CIE-RGB色度空间），101-5
CIE-RGB matching functions（CIE-RGB匹配函数），97-101
CIE-XYZ chromaticity space（CIE-XYZ色度空间），105-8
clipping（裁剪），115
converting between RGB and XYZ（在 RGB 和 XYZ 之间转换），111-13
and CRT displays（CRT显示器），108-11
gamuts（色度范围），113-16
perceivable，generating（可感知的，产生），96-7
primary（原），95
as spectral distributions（光谱分布），88-93
undisplayable（不可显示的），113-16
color-indexing（颜色索引），110
color lookup tables(CLUTs)（颜色查找表(CLUT)），110
color matching functions（颜色匹配函数），97-101
Commission Internationale de L'Eclairage（国际照明协会）
complex polygon（复杂多边形），261
computer aided geometrical design（计算机辅助几何设计（CAGD）），383-433
B-spline curves（B样条曲线），407-15
and Bézier points（Bézier 点），410-11
computation（计算），411-12
control points（控制点），412-13
cubic interpolation（三次插值），425-8
functions（函数），415-18
knot insertion（节点插入），413-14
knot sequences（节点序列），412-13
multiple knot insertion（多节点插入），414-15
piecewise（分段），407
single polynomial segment（单多项式片段），407-8
two-polynomial segment（二多项式片段），408-10
Bézier curves（Bézier 曲线），394-7
application to（应用到），398-9
and Bernstein basis（Bernstein 基），395
in blossoming（在开花中），390-2
converting from general form（从一般形式中转换），402
degree raising（升阶），397-9
polynomial form（多项式形式），394-5
rational curves（有理曲线），400-2
tangent vectors（切向量），395-7
Bézier patches（Bézier 面片），
triangular（三角形的），422-5
blossoming（开花），384、385、387-8
and Bézier curves（Bézier 曲线），390-2
and continuity（连续性），405-6
de Casteljau triangles（de Casteljau 三角形），388-90、392-3
and degenerate polynomials（退化多项式），398
continuity（连续性），402-6
condition（条件），406
geometric（几何的），404-5
parametric（参数化的），403-4
piecewise polynomial curve segments（分段多项式曲线段），402-3
degenerate polynomials and blossoms（退化多项式和

开花)、398
use of (使用)、397-8
polynomials (多项式)、384-7
adaptive forward differences (自适应前向差分)、430-3
affine maps (仿射映射)、385-6
evaluating (评估)、428-33
forward differences (前向差分)、428-30
functions (函数)、385-6
multi-affine maps (多仿射映射)、386-7
surfaces (表面)、418 -19
B-spline (B样条)、422
Bernstein basis (Bernstein 基)、419-20
blossoms for rectangular Bézier patches (矩形Bézier面片的开花)、421-2
parametric (参数化的)、419-20
cone、as primitive (圆锥体、作为基本体素)、379
connection composition (连接合成)、451
conservative visibility algorithms (保守的可见性算法)、490
construction edge (构造边)、321
constructive solid geometry (体素构造表示)、375-82
quadric surfaces (二次曲面)、377-80
ray classification and combination (光线分类和组合)、380-1
continuity in CAGD (CAGD 中的连续性)、402-6
condition (条件)、406
geometric (几何的)、404 -5
parametric (参数化的)、403-4
piecewise polynomial curve segments (分段多项式曲线段)、402-3
control points (控制点)
of B-spline curves (B 样条曲线)、412-13
of Bézier curves (Bézier 曲线)、391
convergence (收敛)、37
convex polygons (凸多边形)、61、163、229
coordinate systems、right and left handed (坐标系统、右手系和左手系)、50
cost、in frame rate control (代价、在帧频控制中)、507
critical surfaces in aspect graph theory (特征图理论中的关键表面)、314
crossover in VEs (虚拟环境中的相似性)、22
cull map (消除图)、502
culling (消除)、489
backface (背面)、491-3
occlusion (遮挡)、493
current transformation matrix (当前的变换矩阵)、209
cylinder、as primitive (圆柱、作为基本体素)、378
CylinderSensor (CylinderSensor)、463

D

de Boor algorithm (de Boor算法)、411-12
de Casteljau triangles (de Casteljau 三角形)、388-90、392-3
degenerate polynomials (退化多项式)、397-8
depth estimation buffer (深度估计缓冲区)、497
depth-sort algorithm (深度排序算法)、247-8
desaturated color (不饱和颜色)、115
desktop virtual reality (桌面虚拟现实)、449
diagonal of multi-affine maps (多仿射映射的对角线)、387
differential areas (差分区域)、59
differential form-factor (差分形状因子)、329
differential solid angle (差分立体角)、57
diffuse radiance (漫反射光亮度)、488
diffuse reflection (漫反射)、133-4、135-8
computing (计算)、138-9
in path tracing (在路径跟踪中)、482
diffuse reflectors (漫反射器)、81
digital differential analyzer (数值差分分析器)、347、468
dimension (维度)、48-50
Dirac delta function (Dirac delta 函数)、91
direct manipulation technique (直接操作技术)、437
direct radiance (直接光亮度)、487
direct specular transmission (直接镜面传导)、149-50
direction of projection (DOP) (投影方向(DOP))、192
directional light sources (线光源)、298
directions (方向)、50-5
and angles (角度)、55-9
over unit sphere (在单位球体上)、55
projected area (投影区域)、58-9
solid angles (立体角)、56-8
spherical coordinate representation (球面坐标表示)、55-6
discontinuity edges、relocation (不连续边、再布置)、324
discontinuity meshing (不连续网格化)、
and radiosity (辐射度)、341
and shadows (阴影)、315-18、324 -5
discontinuity meshing tree (不连续网格树)、321-2
discrete ray tracing (离散光线跟踪)、468-9
distributed ray tracing (分布式光线跟踪)、469-78
aliasing (走样)、470-2
depth of field (景深)、473-7
Monte Carlo (蒙特卡洛)、469-70
motion blur (运动模糊)、477-8

reflection and lighting model（反射和光照模型），472-3
domain（域），385
dominant wavelength （主波长），114
dynamic imposter （动态替代图像），508-10
dynamic scenes，BSP trees in （动态场景，BSP 树中），255-9

E

edge tables （边列表），262-5
EEE surfaces （EEE 表面），315，317，319
emitter events （发射器事件），315
emitters （发射器），137
system，model of（系统，模型），95
EV surfaces （EV表面），315，317，319
exact visibility algorithms （精确的可见性算法），490
exploratory navigation（探索导航），434
extensive display in VE（虚拟环境中大范围显示），436
extremal planes（极值平面），311
extremal shadow boundaries（极值阴影边界），311-14
Eye Coordinates （眼睛坐标），155
eye tracing （视线跟踪），483
eyeball-in-hand navigation （控制眼球的导航），455

F

faces （面），164
fake shadows （伪阴影），300，309-10
field of view （视域），23
fields in VRML97 （在 VRML97中的域），531-2
filtering image space （过滤图像空间），286-7
fish tank virtual reality （鱼缸虚拟现实），449
fixations （固定），27
flat-shaded graphics （平面明暗处理图形），85
flux（通量），328，329
flying vehicle control navigation （飞行媒体控制导航），455
focal planes of lenses（透镜焦点平面），474
focal points of lenses （透镜焦点），473-4
form-factors （形状因子），329
computing （计算），331-5
forward difference evaluation of polynomials （多项式的前向差分估计），428-9，428-33
fovea （凹），27-8
frame buffer （帧缓冲区），110
frame rate （帧频），15
control （控制），507-8
frames （帧），15
FrontRegion clusters （FrontRegion 群），492-3
functions （函数），385

G

gamma correction （gamma 校正），111
gamuts （全色谱），113-16
geometric continuity in CAGD （CAGD中的几何连续性），404-5
geometric realism （几何真实感），8-10
global photon map （全局光子图），486
GLUT system （GLUT系统），223，458-62
Gouraud shading （Gouraud 明暗处理），273-4
graphics（图形），
concepts（概念），128-31
painting metaphor （绘画隐喻）
groups （组），528-30
gulf of evaluation （评估的差距），437
gulf of execution （执行的差距），437

H

half-space （半空间），165
haptic data （触觉数据），23
head-mounted display（HMD） （头盔显示器（HMD）），21，44，436-7
head-movement parallax（头部运动视差），38
hemicube approximation in radiosity （辐射度的半立方体近似值），331-3
hierarchical occlusion maps（HOM） （层次化遮挡图（HOM）），495-7
hierarchical radiosity （层次化辐射度），338-40
hierarchical z-buffer（HZB） （层次化z缓冲区（HZB）），494-5，497
homogeneous space（齐次空间），238-41
Hoppe's algorithm （Hoppe 算法），505
horizontal edges（水平边），263
hue （色调），115
human interaction in virtual environments （在虚拟环境中人的交互），434 -47
body-centered navigation （以身体为中心的导航），444，454
body simulation （人体仿真），439-41
general collision detection （一般的碰撞检测），446-7
locomotion （移动），443-4
object manipulation （对象操作），441-2
object pair collision detection （对象间碰撞检测），444-6
virtual body，interacting with（虚拟人体交互），441-4

virtual reality model （虚拟现实模型），435-9
human-computer interface （人机界面），437-8
immersion（沉浸感），22-6，436-7
and VRML （VRML），447
hyperboloids，as primitives （双曲面，作为基本体素），379

I

iconic representations （图标表示），12-14
illumination，global（光照，全局的），11，465-88
distributed ray tracing （分布式光线跟踪），469-78
aliasing （走样），470-2
depth of field （景深），473-7
Monte Carlo（蒙特卡洛），84，469-70
motion blur（运动模糊），477-8
reflection and lighting model （反射和光照模型），472-3
path tracing （路径跟踪），479-83
photon tracing （光子跟踪），484 -8
density estimation （密度估计），484
KD trees （KD树），485-6
photon maps （光子图），486-8
ray tracing （光线跟踪），465-9
discrete tracing （离散跟踪），468-9
and radiosity （辐射度），483-4
and ray space （光线空间），467-8
illumination realism （光照真实感），10-11
image-based rendering (IBR) （基于图像的渲染 (IBR)），490
image plane（图像平面），38
image point （图像点），122
image precision methods（图像精确方法），244
image space algorithms （图像空间算法），267-96
smooth shading（平滑明暗处理），273-5
Gouraud shading （Gouraud 明暗处理），273-4
Phong shading （Phong 明暗处理），274-5
texturing （纹理生成），275 -93
coordinates，choosing （坐标，选择），287-8
filtering （过滤），286-7
incremental mapping （渐增映射），282-6
mapping （映射），277-82
mipmapping （mipmapping），286-7
OpenGL on （OpenGL ），288-93
texels （纹理像素），276-7
VRML97 on （VRML97 ），293-5
z-buffer visibility algorithm （z缓冲区可见性算法），268-73
polygon fill （多边形填充），270-1
recursive subdivision visibility（递归细分可见性），273
scan-line renderer（扫描线渲染器），269-70
scan-line visibility （扫描线可见性），271-3
image space occlusion in real-time rendering （实时渲染中的图像空间遮挡），493-8
immersion in VE（虚拟环境中的沉浸感），22-6，436-7
immersive virtual environment （沉浸式虚拟环境），22
importance sampling （重要性采样），482
imposters （替代图像），508
impressionism （印象主义），12-14
in-scattering of light （光的入射），75
inclusive display in VE（虚拟环境中的相容显示），436
incorporation （结合），20
incremental texture mapping of image space（图像空间的渐增纹理映射），282-6
intensity of light（光强度），95
interpolators in VRML97 （VRML97 中的内插器），533-5
interpupilary distance (IPD) （瞳孔间距离 (IPD)），217，218
inverse kinematics （逆向运动学），440
irradiance （辐照度），80，328
isotropic objects （均质对象），78

J

jittered sampling （抖动采样），471
Jordan curve （约旦曲线），261

K

Kanizsa's triangle（Kanizsa 三角形），32-3
KD trees in global illumination （全局光照中的 KD 树），485-6
knots （节点），407
Boehm's insertion formula （Boehm 插入公式），414
insertion in B-spline curves （在 B 样条曲线中插入），413-14
multiple knot insertion （多节点插入），414 -15
sequences（序列），412-13

L

Lambert's law （朗伯定律），135-8，273，466
latency in VE （虚拟环境中的延迟），16
layout composition （输出组成），451
level of detail in rendering （渲染中的细节层次），490，503-4
progressive mesh （渐进网格），505-6
static （静态），503-5

Liang-Barsky algorithm （Liang-Barsky 算法），362-4
light buffer （光缓冲区），351
light-facing polygons （朝向光源的多边形），312
light fields in real-time rendering （实时渲染中的光域），511-20
  interpolation （插值），514-17
  representing （表现），517-18
light sources （光源），137
light space （光空间），300
light tracing （光跟踪），483
lighting （光照），73-88
  absorption of （吸收），75
  in graphics （在图形中），130
  isotropic objects （均质对象），78
  radiance （光亮度），
  reflectance （反射），80-1
  in scene description （在场景描述中），530-1
  in three-dimensions （在三维空间中），42
  time invariance of （时间不变性），77
  vacuum （真空），77
  wavelength independence （波长独立），77
lighting model in distributed ray tracing （在分布式光线跟踪中的光照模型），472-3
line equations and intersections （直线方程和交点），355-7
linear perspective in three-dimensions （三维空间中的线性透视），38-40
list priority algorithms （列表优先权算法），244，246-9
  depth-sort （深度排序），247-8
  ordering （排序），246-7
    in object space （在对象空间中），248-9
Local Coordinates （局部坐标），175
local illumination （局部光照），11、133-54
  diffuse reflection （漫反射），135-8
    computing （计算），138-9
  Lambert's law （朗伯定律），135-8
  local specular reflection （局部镜面反射），139-41
  in OpenGL （在 OpenGL 中），150-3
  ray casting in （光线投射），141-3
  ray tracing in （光线跟踪），
    algorithm of （算法），148-50
    direct specular transmission （直接镜面传导），149-50
    direction of reflection （反射方向），148
    perfect specular transmission （完全镜面传导），149
    recursive （递归的），143-8
  in VRML97 （在 VRML97 中），153
local specular reflection （局部镜面反射），139-41
locomotion （移动），434、444
  in real-time interaction （实时交互），454-7
    with 2D device （二维设备），455-6
    scale and accuracy （范围和精度），456
    virtual body，use of （虚拟人体使用），456-7
  in virtual environments （在虚拟环境中），443-4
lumigraph （光照图），512
luminance （光强度），104-5

M

macula （黄斑中心凹），27，93
magnification （放大），277-8
manipulation in real-time interaction （在实时交互中的操作），453-4
  rotation，2D （旋转，二维），454
  translation，2D （平移，二维），453-4
mapping of image space （图像空间的映射），277-82
Markov chain ray tracing （马可夫链光线跟踪），482
Master Coordinates （主坐标），175
mathematics （数学），48-72
  dimension （维），48-50
  directions （方向），50-5
    and angles （角度），55-9
    over unit sphere （在单位球体上），55
    projected area （投影区域），58-9
    solid angles （立体角），56-8
    spherical coordinate representation （球坐标表示），55-6
  flatness-preserving transformations （平面保持变换），59-67
    affine transformations （仿射变换），62-4
    Barycentric combinations （重心组合），59-61
    line segments （线段），60
    standard transformations （标准变换），64-7
  points （点），50-1
  position （位置），50-5
  quaternians （四元数），67-72
    addition （加法），68
    definition （定义），67-8
    polar representation of （极坐标表示），70-1
    quaternian inverse （四元数的逆），69-70
    quaternian multiplication （四元数乘法），69
    rotation （旋转），71-2
    scalar multiplication （数乘），68-9
  vectors （矢量），50-1
    addition （加法），51-2
    cross product （叉乘），53-5

inner product of two (内积), 53
norm of (模, 范数), 52-3
scaling (缩放), 52
merge composition (合并组成), 451
mesh decimation (网格粗化), 506
meshing (网格化),
and radiosity (辐射度), 337-41
adaptive subdivision (自适应细分), 337-8
discontinuity (不连续), 341
hierarchical radiosity (层次化辐射度), 338-40
uniform (一致), 337
on shadows (在阴影上),
constructing on faces (面上构造), 321-2
discontinuity (不连续性), 315-18, 324-5
vertices, illuminating (顶点, 光照), 322-4
metamers (条件等色), 91, 96
mind's eye (想像力), 32
minification (缩小), 277-8
mipmapping (mipmapping), 286-7
MixedRegion clusters (MixedRegion 群), 493
modality, avoiding in real-time interaction (实时交互中的通道屏蔽), 457-8
virtual widgets (虚拟小构件), 458
Modeling Coordinates (建模坐标), 175
monochromatic energy (单色能量), 91
Monte Carlo Path Tracing (蒙特卡洛路径跟踪), 84, 85-6
Monte Carlo Photon Tracing (蒙特卡洛光子跟踪), 84, 87, 484
Monte Carlo ray tracing (蒙特卡洛光线跟踪), 469-70, 482
motion blur in global illumination (全局光照中的运动模糊), 477-8
motion parallax (运动视差), 38
multi-affine maps in CAGD (在CAGD中的多仿射映射), 386-7

N

Necker cube (Necker 立方体), 14
negative half-space (负半空间), 165, 355
negative horizontal parallax (负水平线视差), 217
negative subspaces (负子空间), 305
Nicholl-Lee-Nicholl clipping algorithm (Nicholl-Lee-Nicholl裁剪算法), 364-5
non-emitter events (非发射器事件), 315
non-immersive systems (非沉浸式系统), 435
non-immersive virtual reality (非沉浸式虚拟现实), 449
non-participating medium for light transport (光传导的非参与性媒体), 77
non-uniform space subdivision (非一致空间细分), 348-50
BSP trees (BSP 树), 349-50
octrees (八叉树), 348-9
non-zero winding number rule (非零匝数规则), 261-2
normalizing vectors (规则化矢量), 52
normals, interpolation of (法向, 插值), 275

O

Object Coordinates (对象坐标), 175
object manipulation in VEs (虚拟环境中的对象操作), 442-3
object pair collision detection (对象间碰撞检测), 444-6
object precision methods (对象精确方法), 243-4
object space occlusion (对象空间遮挡), 498-9
object space ordering of polygons (多边形的对象空间排序), 248-9
objects in virtual environments (虚拟环境中的对象), 3-5
occluder shadows (遮光板阴影), 501-2
occluders (遮光板), 298
occlusions (遮挡),
cells and portals method (单元和入口方法), 500
culling (消除), 493-4
in three-dimensions (在三维空间中), 41-2
octrees (八叉树), 348-9
odd-even rule in rendering polygons (渲染多边形中的奇偶规则), 261
opacity threshold (不透明阈值), 497
Open Inventor (Open Inventor), 458
OpenGL (OpenGL),
commands (指令),
GL_ACCUM, 522
GL_ACCUM_BUFFER_BIT, 522
GL_AMBIENT, 151, 152, 214
GL_AUTO_NORMAL, 227
GL_BLEND, 151, 152, 214
GL_CLAMP, 291
GL_COLOR_BUFFER_BIT, 221, 227, 522
GL_ DECAL, 291
GL_DEPTH_BUFFER_BIT, 221, 227, 522
GL_DEPTH_TEST, 152, 222, 228
GL_DIFFUSE, 151, 152, 214, 228
GL_FLAT, 151, 152, 222, 227
GL_FRONT, 152, 214, 228
GL_LIGHT0, 152, 228
GL_LIGHTING, 152, 228

GL_LINEAR，291
GL_MODELVIEW，180，212，215，225，227，292，521，522
GL_MODULATE，291
GL_NEAREST，291
GL_NORMALIZE，152，227
GL_POLYGON，180，214，221，225，291，293
GL_POSITION，152，228
GL_PROJECTION，213，227，521，522
GL_RETURN，522
GL_RGB，291
GL_SHININESS，151，152，214
GL_SMOOTH，151，152，222
GL_SPECULAR，151，152，214
GL_TEXTURE_2D，291
GL_TEXTURE_ENV，291
GL_TEXTURE_ENV_MODE，291
GL_TEXTURE_MAG_FILTER，291
GL_TEXTURE_MIN_FILTER，291
GL_TEXTURE_WRAP_S，291
GL_UNPACK_ALIGNMENT，290
GL_UNSIGNED_BYTE，291
GL_VIEWPORT，521，522
glAccum，522
glBegin，180，214，221，225，291，293
glClear 221，227，522
glClearColor，152，222，228
glClearDepth，152，222，228
glColor3f，221
glDisable，152，214
GLdouble，211，212，213，215，216，227，521，522
glEnable，151，152，153，214，222，227，228，291
glEnd，180，214，221，225，292，293
GLenum，151
GLfloat，151，152，211，227
glFlush，221，293，522
glFrustrum，522
glFrustum，212，213，521
glGetIntegerv，521，522
GLint，290，460，521
glLightfv，153，228
glLoadIdentity，211，213，227，228，521，522
glLoadMatrix，212
glLoadMatrixd，211
glMaterial，151
glMaterialfV，151，152，214，228
glMatrixMode，179，180，211，212，213，215，225，227，228，292，521，522
glMultMatrixd，215
glNormal3d，214
glPixelStorei，290
glPopMatrix，181，215，226，227，292，293
glPushMatrix，180，181，215，226，227，292，293
glRotate，179
glRotated，179，180，181，226，227，292
glScaled，181，226，293
glShadeModel，152，214，222，227
GLsizei，227，291
glTexCoord2f，291，292，293
glTexEnvf，291
glTexImage2D，291
glTexParameterf，291
glTranslate，179
glTranslated，179，180，181，211，212，213，226，292，293
GLubyte，289，290
glVertex3d，180，214，225，291，292，293
glVertex3f，221
glViewport，222，227
filtering in（在……中过滤），286
in local illumination（在局部光照中），150-3
rendering object hierarchy（渲染对象层次结构），224-8
scene construction using（使用……进行场景构造），178-81
on texturing image space（在纹理图像空间上），288-93
viewing in（在……中观察），210-17
OpenGL Utility Library（GLU）（OpenGL实用程序库（GLU））
gluLookAt，216，228
gluPerspective，216，227，521，522
OpenGL Utility Toolkit (GLUT) (OpenGL实用程序包(GLUT))
GLUT_DEPTH，222，228
GLUT_DOUBLE，228
GLUT_DOWN，460
GLUT_KEY_DOWN，460
GLUT_KEY_UP，460
GLUT_LEFT_BUTTON，460
GLUT_RGBA，222，228
GLUT_RIGHT_BUTTON，459
glutAddMenuEntry，459
glutAttachMenu，459
glutCreateMenu，459

glutCreateWindow，223，228
glutDisplayFunc，216，223，228
glutIdleFunc，216，228，459
glutInit，222，228
glutInitDisplayMode，222，228
glutInitWindowSize，222，228
glutKeyboardFunc，459
glutKeyboardUpFunc，459
glutMainLoop，223，228
glutMotionFunc，459
glutMouseFunc，459
glutPostRedisplay，460，461
glutReshapeFunc，216，223，228
glutSetWindow，223，228
glutSpecialFunc，459
glutSpecialUpFunc，459
glutSwapBuffers，227，522
optic nerve（视神经），27-8
orthographic parallel projection（正交平行投影），195
Oslo algorithm（Oslo算法），414-15
out-scattering of light（光的出射），75

P

Painters' Algorithm（画家算法），244
painting metaphor（绘画隐喻），119-32
forming displayed image（形成显示图像），126-7
ray casting（光线投射），124-6
scene（场景），122-3
simple camera（简单照相机），123-4
terminology（术语），122
paraboloid，as primitive（抛物面，作为基本体素），379
parallax effect（视差效果），26
parallel projection（平行投影），195-6
parametric continuity in CAGD（在CAGD中的参数化连续性），403-4
parametric line clipping（参数化线裁剪），362-4
parametric surfaces in CADG（在CADG中的参数化表面），419-20
paraxial rays（轴旁光线），475
participating medium for light transport（光传送的参与性媒体），77
partitioning planes（分割平面），250
patches（面片），337
energy between two（二者之间的能量），328-30
rectangular（矩形），418
path tracing in global illumination（全局光照中的路径跟踪），479-83
penumbras（半影），298，310-26
analytical determination methods（解析确定方法），310，311-25
aspect graphs（特征图），314-15
discontinuities，locating（不连续，位于），318-20
discontinuity meshing（不连续网格化），315-18，324-5
extremal shadow boundaries（极值阴影边界），311-14
mesh，constructing on faces（网格，在面上构造），321-2
mesh vertices，illuminating（网格顶点，光照），322-4
sampling methods（采样方法），311，325-6
perfect specular transmission（完全镜面传导），134，149
persistence，in CRT displays（持续，在CRT显示器中），109
perspective projection（透视投影），196-7
Phong lighting model（Phong光照模型），140
Phong shading（Phong明暗处理），274-5
photo-realistic images（相片逼真的图像），11
photometry（光度测定），89，104
photon maps（光子图），485，486-8
photon tracing（光子跟踪），484-8
density estimation（密度估计），484
KD trees（KD树），485-6
photon maps（光子图），486-8
photons（光子），74
photopic vision（适应光的视觉），93
Pinch Glove（Pinch Glove），442
pixel coherence（像素相关性），262
pixels，in CRT（像素，在CRT中），108，122，125
plane distances，transforming in projection（平面距离，在投影中转换），201-2
planes（平面），
as primitives（作为基本体素），378
in scene construction（在场景构造中），163-8
equation（方程，等式），163-5
point light sources（点光源），298
points（点），50-1
polar representation of quaternians（四元数的极坐标表示），70-1
Polhemus Fastrak（Polhemus Fastrak），440，451
polygon fill（多边形填充），270-1
polygons（多边形），
clipping（裁剪），229-42
in canonical viewing space（在规范观察空间中），236-8
in 3D（在三维中），234-41

in homogeneous space (在齐次空间中), 238-41
in projection space (在投影空间中), 235-6
Sutherland-Hodgman algorithm (2D) (Sutherland-Hodgman 算法(二维)), 230-2
Weiler-Atherton algorithm (Weiler-Atherton 算法), 232-4
convex (凸的), 61
intersecting by rays (与光线相交), 165-8
rasterization (光栅化), 261-4
coherence, exploiting (相关性, 利用), 262
edge tables (边表), 262-4
inside polygon (在多边形内), 261-2
rendering (渲染), 260-5
in scene construction (在场景构造中), 163-8
polyhedra(多面体)
in scene construction (在场景构造中), 168-73
vertex-face data structure for (顶点-面数据结构), 168-70
winged-edge data structure for (翼边数据结构), 170-2, 184-8
polynomials in CAGD (CAGD 中的多项式), 384-7
affine maps (仿射映射), 385-6
degenerate (退化),
and blossoms (开花), 397-8
use of (使用), 397-8
evaluating (评估), 428-33
adaptive forward differences (自适应前向差分), 430-3
forward differences (前向差分), 428-9
functions (函数), 385-6
multi-affine maps (多仿射映射), 386-7
portals occlusion (入口遮挡), 500
position (位置), 50-5
positive half-space (正半空间), 165, 355
positive horizontal parallax (正水平视差), 217
positive subspaces (正子空间), 305
potentially visible set (潜在的可见集合), 491
presence in VE (在虚拟环境中存在), 17-22, 44, 435
primary colors (原色), 95
principal vanishing point (主灭点), 192
priority list algorithms (优先权列表算法), 260
procedural texture (程序化纹理), 276
progressive meshes (渐进网格), 505-6
progressive refinement algorithm (渐进细化算法), 335-6
projection (投影),
in camera model (在照相机模型中), 194-202
canonical frames (规范框架), 197-200
canonical projection space (规范投影空间), 200-1
clipping front and back planes (裁剪前平面和后平面), 193-4, 201
parallel (平行), 195-6
perspective (透视), 196-7
plane distances, transforming (平面距离, 转换), 201-2
type of (类型), 192
in graphics (在图形中), 130
projection plane (投影平面), 122
projection space (投影空间), 198, 200, 235-6
ordering in (在……中的排序), 246-7
rendering and texturing (渲染和纹理生成), 268
proprioception (本体感受), 23

Q

quadric solids (二次实体), 377-8
quadric surfaces (二次曲面), 377-80
quadtrees (四叉树), 349
quaternians (四元数), 67-72
addition (加法), 68
definition (定义), 67-8
polar representation of (极坐标表示), 70-1
quaternian inverse (四元数的逆), 69-70
quaternian multiplication (四元数乘法), 69
rotation (旋转), 71-2
scalar multiplication (数乘), 68-9

R

radiance (光亮度), 78-80, 135
equation (方程), 81-3, 512
in graphics (在图形中), 131
solutions (解), 83-7
flat-shaded graphics (平淡明暗处理图形), 85
Monte Carlo Path Tracing (蒙特卡洛路径跟踪), 85-6
Monte Carlo Photon Tracing (蒙特卡洛光子跟踪), 87
and radiosity (辐射度), 86-7
ray tracing (光线跟踪), 85
in real-time graphics (在实时图形中), 86
reflected (反射), 80
and visibility (可见性), 87
radiance spectral distribution (光亮度光谱分布), 104
radiant energy (辐射能量), 75
radiant intensity (光强度), 79
radiosity (辐射度), 80, 134, 327-41
adaptive texture (rex) (自适应纹理 (rex)), 483

definition（定义），328
energy between two patches（两面片之间的能量），328-30
equations（方程），330-1
form-factors, computing（形状因子，计算），331-5
hemicube approximation（半立方体逼近），331-3
Nusselt analog（Nusselt类比），331-3
with ray casting（光线投射），334-5
and lighting（光照），86-7
meshing（网格化），33 7-41
adaptive subdivision（自适应细分），337-8
discontinuity（不连续），341
hierarchical radiosity（层次化辐射度），338-40
uniform（一致），337
progressive refinement method（渐进细化方法），335-6
algorithm（算法），335-6
ambient term（环境项），336
and ray tracing in global illumination（全局光照中的光线跟踪），483-4
rendering（渲染），341
raster scanning, in CRT（光栅扫描，在CRT中），109
rational curves（有理曲线），400-2
rational polynomials（有理多项式），400
ray（光线），79
ray casting（光线投射），
in graphics（在图形中），124 -6
in local illumination（在局部光照中），141-3
radiosity form-factors, computing with（辐射度形状因子，计算），334 -5
ray coherence methods in ray tracing（光线跟踪中的光线相关性方法），350-3
light buffer（光缓冲区），351
ray tracing（光线跟踪），134，343-53
bounding volumes（包围体），344 -5
hierarchical（层次化），345-6
selection of（选择），346-7
distributed（分布式），469-78
aliasing（走样），470-2
depth of field（景深），473-7
Monte Carlo（蒙特卡洛），469-70
motion blur（运动模糊），477-8
reflection and lighting model（反射和光照模型），472-3
in global illumination（在局部光照中），84，465-9
discrete tracing（离散跟踪），468-9
and radiosity（辐射度），84，483-4
and ray space（光线空间），467-8
intersection calculations（相交计算），344
in local illumination（在局部光照中），
algorithm of（算法），148-50
direct specular transmission（直接镜面传导），149-50
direction of reflection（反射方向），148
perfect specular transmission（完全镜面传导），149
recursive（递归），143 -8
non-uniform space subdivision（非一致空间细分），348-50
BSP trees（BSP树），349-50
octrees（八叉树），348-9
and radiance（光亮度），85
ray coherence methods（光线相关性方法），350-3
light buffer（光缓冲区），351
ray classification（光线分类），351-3
slow speed of（慢速度），190
uniform space subdivision（一致空间细分），346-8
real-time（实时），14-17
interaction in shared VE（在共享虚拟环境中的交互），16-17
object interaction in（对象交互），16
real-time interaction（实时交互），449-64
desktop devices（桌面设备），450-2
locomotion（移动），454-7
with 2D device（二维设备），455-6
scale and accuracy（范围和精度），456
virtual body, use of（虚拟人体使用），456-7
manipulation（操作），453-4
rotation, 2D（旋转，二维），454
modality, avoiding（通道屏蔽），457-8
virtual widgets（虚拟小构件），458
selection（选择），452-3
translation, 2D（平移，二维），453-4
in VRML（在VRML中），462-4
sensors（传感器），463-4
vehicles（媒体），462-3
real-time rendering process（实时渲染程序），489-523
anti-aliasing（反走样），522-2
image-based（基于图像的），508-11
light fields（光域），511-20
interpolation（插值），514 -17
representing（表现），517-18
multi-resolution representations（多分辨率表示），503-8
frame rate control（帧频控制），507-8
mesh decimation（网格粗化），506

progressive meshes（渐进网格），505-6
static level of detail（静态细节层次），503-5
visibility processing（可见性处理），490-503
architectural scenes（建筑场景），500-3
backface culling（背面消除），491-3
image space occlusion（图像空间遮挡），493-8
object space occlusion（对象空间遮挡），498-9
occluders, selecting（遮光板，选择），499-500
occlusion culling（遮挡消除），493
view volume（视景体），491-3
VRML on（VRML），522-3
real-time walkthrough（实时漫游），15-16
realism（真实感），7-17
behavioral（行为的），11-12
geometric（几何的），8-10
iconic representations（图标表示），12-14
illumination（光照），10-11
and real-time（实时），14-17
interaction in shared VE（在共享虚拟环境中的交互），16-17
object interaction（对象交互），16
real-time walkthrough（实时漫游），15-16
reality and virtual environments（真实和虚拟环境），26-36
receivers（接收器，接收表面），298
rectangular patches（矩形面片），418
rectilinear surfaces（直纹面），418
recursive ray tracing（递归光线跟踪），143-8
recursive subdivision visibility algorithm（递归细分可见性算法），273
reflectance（反射），80-1
reflected radiance（反射光亮度），80
reflection（反射），
in global illumination（在全局光照中），472-3
local specular（局部镜面），139-41
total internal（完全内部的），149
regularized set operations（规则化集合操作），376
rendering process（渲染过程），128
equation for light intensity（光强度方程），479
object hierarchy in OpenGL（在OpenGL中的对象层次结构），224-8
on polygons（在多边形上），260-5
and radiosity（辐射度），341
rasterization（光栅化），261-5
coherence, exploiting（相关性，利用），262
edge tables（边表），262-4
inside polygon（在多边形内），261-2
real-time, *see* real-time rendering process（实时，参见实时渲染过程）
scan-line in z-buffer visibility algorithm（z缓冲区可见性算法中的扫描线），269-70
resolution, in CRT displays（分辨率，在CRT显示器中），109
retina（视网膜），27，93
rotation（旋转），
of quaternians（四元数），71-2
in standard transformation matrix（在标准变换矩阵中），65-7

S

saccades（快速扫描），27-8
saccadic suppression（快速扫描抑制），42
sampling（采样），366
of penumbras（半影），311，325-6
vision as（视觉），27-8
saturated color（饱和颜色），115
scaling matrix in standard transformations（标准变换中的缩放矩阵），65
scan-line coherence（扫描线相关性），262
scan-line renderer（扫描线渲染器），269-70
scanpath（扫描路径），28-32
scene（场景），
hierarchy（层次结构），173-8
concepts（概念），173-6
data structures（数据结构），177-8
matrices associated with objects（与对象关联的矩阵），176-7
in painting metaphor（在绘画隐喻中），122-3
planes（平面），163-8
equation（方程），163-5
polygons（多边形），163-8
intersecting by rays（与光线相交），165-8
polyhedra（多面体），168-73
vertex-face data structure for（顶点-面数据结构），168-70
winged-edge data structure for（翼边数据结构），170-2，184-8
using OpenGL（使用OpenGL），178-81
using VRML97（使用VRML97），181-2
scene description in VRML97（VRML97中的场景描述），526-31
appearance（外观），528
groups, transformations and scene graphds（组、变换和场景图），528-30
lights（光），530-1
shapes and geometry（形状和几何），528

scene graphs （场景图），528-30
scene-in-hand navigation （所掌握的场景导航），455
scotopic vision （暗视视觉），93
second degree Bézier curve （二阶Bézier 曲线），391
selection in real-time interaction（实时交互中的选择），452-3
sensors in real-time interaction（实时交互中的传感器），463-4
separating plane test （分割平面测试），445
separation of scene specification in graphics （图形中场景描述的分离），128
set operations （集合操作），375-7
shading in three-dimensions （三维中的明暗处理），41
shadow depth buffer（阴影深度缓冲区），300
shadow feelers （阴影感知器），146，484
shadow frustum （阴影锥），498
shadow map （阴影图），300
shadow planes （阴影平面），301-4
shadow testing（阴影测试），351
shadow volume BSP (SVBSP) trees（阴影体BSP(SVBSP)树），299，301，304-9，499
shadow volume (SV) of polygon（多边形的阴影体(SV)），301-2
shadows （阴影），297-326
  penumbras （半影），310-26
    analytical determination methods （解析确定方法），310，311-25
    aspect graphs（特征图），314 -15
    discontinuities，locating （不连续，位于），318-20
    discontinuity meshing （不连续网格化），315 -18，324-5
    extremal shadow boundaries（极值阴影边界），311-14
    mesh，constructing on faces（网格，在面上构造），321-2
    mesh vertices，illuminating（网格顶点，光照），322-4
    sampling methods （采样方法），311，325-6
  sharp（硬的），298
  soft （柔和的），298
  in three-dimensions（在三维中），41
  umbras（本影），299-310
    BSP trees （BSP 树），304-9
    extremal shadow boundaries （极值阴影边界），311-14
    fake shadows （伪阴影），309-10
    volumes （体），300-4
shininess （发光参数），140-1
simple polygon （简单的多边形），229，261
simple virtual camera （简单的虚拟照相机），123-4
size constancy scaling （尺寸固定的缩放），35
smooth shading of image space（图像空间的平滑明暗处理），273-5
  Gouraud shading （Gouraud 明暗处理），273-4
  Phong shading（Phong 明暗处理），274 -5
Snell's law （Snell定律），149，466
solid angles （立体角），56-8
space（空间），42-3
space coherence （空间相关性），344
Spaceball（空间球），451
spectral luminous efficiency curve（光谱发光效率曲线），104
spectral radiant power distribution （光谱辐射能量分布），90
spectrophotometer（分光光度计），90
specular radiance（镜面光亮度），487
specular reflection（镜面反射），81，133，140
  in path tracing （在路径跟踪中），482
specular transmission （镜面传导），
  direct（直接的），149-50
  perfect（完全的），149
sphere，as primitive （球体，作为基本体素），378
SphereSensor（SphereSensor），463，464
spherical coordinate representation （球坐标表示），55-6
standard transformations （标准变换），64-7
    composition of（组成），66
    rotation matrix（旋转矩阵），65-7
    scaling matrix（缩放矩阵），65
    translation matrix （平移矩阵），64
stencil buffer（模板缓冲区），303
stereo views（立体视图），
  with camera model（照相机模型），219-24
  creating （创建），217-24
  setting up （建立），217-19
stratified sampling（分层采样），471
streaming of light （光流），75
supersampling（超采样），471，520
surfaces in CADG （在CADG中的表面），418-19
  B-spline（B样条），422
  Bernstein basis （Bernstein 基），419-20
  blossoms for rectangular Bézier patches（矩形 Bézier 面片的开花），421-2
  parametric （参数化），419-20
Sutherland-Hodgman algorithm (2D) （Sutherland-Hodgman 算法(二维)），230-2，234-5

symmetry of multi-affine maps （多仿射映射的对称），387

T

telepresence in VE（虚拟环境中的遥现），435
tensor product surfaces （张量积表面），418，419
texels （纹理像素），276-7
texture gradient in three-dimensions （三维中的纹理梯度），40-1
texture mapping （纹理映射），275
texturing image space（纹理图像空间），275-93
coordinates，choosing （坐标，选择），287-8
filtering （过滤），286-7
incremental mapping（渐增映射），282-6
mapping （映射），277-82
mipmapping（mipmapping），286-7
OpenGL on （OpenGL），288-93
texels （纹理像素），276-7
VRML97 on （VRML97），293-5
three-dimensions （三维），
clipping polygons in （在……中裁剪多边形），234-41
lighting （光照），42
linear perspective （线性透视），38-40
occlusions （遮挡），41-2
shadows and shading （阴影和明暗处理），41
texture gradient （纹理梯度），40-1
timesensors in VRML97（VRML97中的时间传感器），533-5
torus，as primitive （圆环面，作为基本体素），379
total internal reflection （总的内部反射），149
TouchSensor（TouchSensor），463
tracking in VE （在虚拟环境中跟踪），23
translation matrix in standard transformations （标准变换中的平移矩阵），64
transparent objects，ray tracing in （透明对象，光线跟踪），146-8
triangular Bézier patches in CADG （CADG 中的三角形Bézier面片），422-5
true color system （真彩色系统），110
true texture coordinates （真纹理坐标），284
two-line parameterization （两直线参数化），512
two-plane parameterization （两平面参数化），513

U

umbras （本影），298，299-310
BSP trees （BSP 树），304-9
extremal shadow boundaries （极值阴影边界），311-14
fake shadows （伪阴影），309-10
volumes （体），300-4
undisplayable colors （不可显示的颜色），113-16
uniform knot sequence （一致节点序列），407
uniform space subdivision （一致空间细分），346-8
clipping line segments（裁剪线段），370-4
unshot radiosity （未击中辐射度），335
UVN coordinate system （UVN 坐标系统），156

V

vacuum （真空），77
vanishing point （灭点），192
vectors （矢量），50-1
addition （加法），51-2
cross product （叉乘），53-5
inner product of two（两向量的内积），53
norm of （模，范数），52-3
scaling（缩放），52
vehicles in real-time interaction （实时交互中的媒体工具），462-3
vertex-face data structure for polyhedra（多面体的顶点-面数据结构），168 -70
vertical parallax （垂直视差），217
vertices of polygons（多边形顶点），229
view-dependence （视图依赖），83-4
view-independence （视图独立），83-4
view plane （视平面），38，122
View Plane Distance (VPD) （视图平面距离 (VPD)），192
View Plane Normal (VPN)（视图平面法向 (VPN)），123，156，191，192，196
View Plane Window (VPW) （视图平面窗口 (VPW)），122，123，193
View Reference Point (VRP)（视图参考点 (VRP)），156，191，192，196
view up Vector (VUV) （视图上方矢量 (VUV)），156，191，192，196
view volume （视景体），
in graphics （在图形中），128-9
in real-time rendering （在实时渲染中），491-3
Viewing Coordinates （观察坐标），155
viewing graphics （观察图形），128
virtual body，interacting with（虚拟人体交互），441-4
body-centered navigation （以身体为中心的导航），444，454
locomotion （移动），443-4
object manipulation in （对象操作），441-4
virtual devices （虚拟设备），451

virtual environments (虚拟环境), 3-7
constancy in time and space (时间和空间不变性), 42-3
content of (内容), 5-7
human interaction in (人的交互), 434-47
body-centered navigation (以身体为中心的导航), 444, 454
body simulation (人体仿真), 439-41
general collision detection (一般的碰撞检测), 446-7
human-computer interface (人机界面), 437-8
immersion (沉浸感), 22-6, 436-7
locomotion (移动), 443-4
object manipulation (对象操作), 442-3
object pair collision detection (对象间碰撞检测), 444-6
virtual body, interacting with (虚拟人体交互), 441-4
virtual reality model (虚拟现实模型), 435-9
and VRML (VRML), 447
hypotheses of visual mind (视觉记忆假设), 43-7
real-time interaction in (实时交互), 16-17
real-time rendering in (实时渲染), 489-523
anti-aliasing (反走样), 520-2
architectural scenes (建筑场景), 500-3
backface culling (背面消除), 491-3
image-based (基于图像的), 508-11
image space occlusion (图像空间遮挡), 493-8
light fields (光域), 511-20
multi-resolution representations (多分辨率表示), 503-8
object space occlusion (对象空间遮挡), 498-9
occluders, selecting (遮光板, 选择), 499-500
occlusion culling (遮挡消除), 493
view volume (视景体), 491-3
visibility processing (可见性处理), 490-503
VRML on (VRML), 522-3
and reality (真实), 26-36
three-dimensions (三维), 36-42
lighting (光照), 42
linear perspective (线性透视), 38-40
occlusions (遮挡), 41-2
shadows and shading (阴影和明暗处理), 41
texture gradient (纹理梯度), 40-1
workings of (工作), 26-47
virtual prototyping (虚拟原型), 10
virtual reality (虚拟现实), 26-36
model (模型), 435-9
virtual widgets (虚拟小构件), 458
visibility (可见性), 243-59
algorithms (算法), 490
back-face elimination (背面删除), 244-6
binary space partition trees (二叉空间分割树), 250-9
constructing (构造), 250-5
in dynamic scenes (在动态场景中), 255-9
rendering (渲染), 253
cell-to-cell (单元到单元), 501
culling (消除), 489
list priority algorithms (列表优先权算法), 246-9
depth-sort (深度排序), 247-8
ordering (排序), 246-7, 248-9
processing in real-time rendering (实时渲染中的处理), 490-503
architectural scenes (建筑场景), 500-3
backface culling (背面消除), 491-3
image space occlusion (图像空间遮挡), 493-8
object space occlusion (对象空间遮挡), 498-9
occluders, selecting (遮光板, 选择), 499-500
occlusion culling (遮挡消除), 493
view volume (视景体), 491-3
and radiance (光亮度), 87
visible surface determination (可见表面确定), 243, 299
vision, sampling process (视觉, 采样过程), 27-8
visual events in aspect graph theory (特征图理论中的视觉事件), 314
visual mind, hypotheses of (视觉记忆, 假设), 43-7
visual system, model of (视觉系统, 模型), 93-4
vivid display in VE (虚拟环境中的生动显示), 436
voxels (体素), 347
VPL dataglove (VPL 数据手套), 452
VRML97 (VRML97),
as animation description (动画描述), 531-5
abstract data flow (抽象数据流), 531
data flow and routes (数据流和路径), 532-3
fields (域), 531-2
timesensors and interpolators (时间传感器和内插器), 533-5
camera, generalizing in (照相机, 一般化), 160
human interaction (人的交互), 447
as interactive experience (交互经验), 535-6
in local illumination (在局部光照中), 153
in real-time interaction (在实时交互中), 462-4
sensors (传感器), 463-4
vehicles (媒体工具), 462-3
on real-time rendering (在实时渲染上), 522-3
scene construction using (使用……的场景构造),

181-3
as scene description （场景描述），526-31
appearance （外观），528
groups, transformations and scene graphs（组、变换和场景图），528-30
lights （光），530-1
nodes and fields（节点和域），526-7
shapes and geometry（形状和几何），528
scripting （脚本构造），535-6
on texturing image space （在纹理图像空间上），293-5

W

walkthrough （漫游），434
real-time（实时），15-16
wedges （楔形），319
Weiler-Atherton algorithm （Weiler-Atherton 算法），232-4
winged-edge data structure for polyhedra（多面体的翼边数据结构），170-2
C specification of （C语言描述），184-8
World Coordinates (WC) （世界坐标(WC)），123，155，177

Z

z-buffer visibility algorithm（z缓冲区可见性算法），268-73
polygon fill （多边形填充），270-1
recursive subdivision visibility algorithm （递归细分可见性算法），273
scan-line renderer （扫描线渲染器），269-70
scan-line visibility （扫描线可见性），271-3
z-pyramid （z棱锥），494